龙芯 GS464 处理核手册

中国科学院计算技术研究所

北京龙芯中科技术服务中心有限公司

中文版本： 版本 1.01，二零一零年十一月

目录

插图

表格

第一章　结构概述

GS464 是一款实现 64 位 MIPS64 指令集的通用 RISC 处理器 IP。其基本结构如下图所示。

图 1-1 GS464 处理器核的基本结构

GS464 的指令流水线每个时钟周期取四条指令进行译码，并且动态地发射到五个全流水的功能部件中。虽然指令在保证依赖关系的前提下进行乱序执行，但指令的提交还是按照程序原来的顺序以保证精确例外和访存顺序执行。

四发射的超标量结构使得指令流水线中指令和数据相关问题十分突出，GS464 采用乱序执行技术和激进的存储系统设计来提高流水线的效率。乱序执行技术包括寄存器重命名技术、动态调度技术和转移预测技术。寄存器重命名解决 WAR（读后写）和 WAW（写后写）相关，并用于例外和错误转移预测引起的精确现场恢复，GS464 分别通过 64 项的物理寄存器堆进行定点和浮点寄存器的重命名。动态调度根据指令操作数准备好的次序而不是指令在程序中出现的次序来执行指令，减少了 RAW（写后读）相关引起的阻塞，GS464 有一个 16 项的定点保留站和一个 16 项的浮点保留站用于乱序发射，并通过一个 64 项的 Reorder 队列（简称 ROQ）实现乱序执行的指令按照程序的次序提交。转移预测通过预测转移指令是否成功跳转来减少由于控制相关引起的阻塞，GS464 使用 16 项的转移目标地址缓冲器（Branch Target Buffer，简称 BTB），2K 项的转移历史表（Branch History Table，简称 BHT）9 位的全局历史寄存器，（Global Histiry Registor，简称 GHR），和 4 项的返回地址栈（Return Address Stack，简称 RAS）进行转移预测。

GS464 先进的存储系统设计可以有效地提高流水线的效率。GS464 的一级 Cache 由 64KB 的指令 Cache 和 64KB 的数据 Cache 组成，均采用四路组相联的结构。GS464 的 TLB 有 64 项，采用全相联结构，每项可以映射一个奇页和一个偶页，页大小在 4KB 到 16MB 之间可变。GS464 通过 24 项的访存队列以及 8 项的访存失效队列（Miss Queue）来动态地解决地址依赖，实现访存操作的乱序执行、非阻塞 Cache、取数指令猜测执行（Load Speculation）等访存优化技术。GS464 支持 128 位的访存操作，其虚地址和物理地址均为 48 位。

GS464 有两个定点功能部件和两个浮点功能部件。每个浮点部件都可以全流水地执行 64 位双精度的浮点乘加操作，并通过浮点指令的 fmt 域的扩展执行 32 位和 64 位的定点指令，以及 8 位和 16 位的用于媒体加速的 SIMD 指令。

GS464 支持 MIPS 公司的 EJTAG 调试规范，采用标准的 AXI 接口，其指令 Cache 实现了奇偶校验，数据 Cache 实现了 ECC 校验。上述特点可以增加龙芯处理器的适用范围。

1.1　流水线

GS464 的基本流水线包括取指、预译码、译码、寄存器重命名、调度、发射、读寄存器、执行、提交等 9 级，每一级流水包括如下操作。

- **取指 (Instruction Fetch) 流水级：** 用程序计数器 PC 的值去访问指令 Cache 和指令 TLB，如果指令 Cache 和指令 TLB 都命中，则把四条新的指令取到指令寄存器 IR。
- **预译码 (Pre-decoder) 流水级：** 主要对转移指令进行译码并预测跳转的方向。译码流水级把 IR 中的四条指令转换成龙芯的内部指令格式送往寄存器重命名模块。
- **寄存器重命名 (Register remapper) 流水级：** 为逻辑目标寄存器分配一个新的物理寄存器，并将逻辑源寄存器映射到最近分配给该逻辑寄存器的物理寄存器。
- **调度 (Schedule) 流水级：** 将重命名的指令分配到定点或浮点保留站中等待执行，同时送到 ROQ 中用于执行后的顺序提交；此外，转移指令和访存指令还分别被送往转移队列和访存队列。
- **发射流水级：** 从定点或浮点保留站中为每个功能部件选出一条所有操作数都准备好的指令；在重命名时操作数没准备好的指令，通过侦听结果总线和 forward 总线等待它的操作数准备好。
- **读寄存器流水级：** 为发射的指令从物理寄存器堆中读取相应的源操作数送到相应的功能部件。
- **执行 (Execution) 流水级：** 根据指令的类型执行指令并把计算结果写回寄存器堆；结果总线还送往保留站和寄存器重命名表，通知相应的寄存器值已经可以使用了。
- **提交 (Submit) 流水级：** 按照 Reorder 队列记录的程序的顺序提交已经执行完的指令，GS464 最多每拍可以提交四条指令，提交的指令送往寄存器重命名表用于确认它的目的寄存器的重命名关系并释放原来分配给同一逻辑寄存器的物理寄存器，并送往访存队列允许那些提交的存数指令写入 Cache 或内存。

上述是基本指令的流水级，对于一些较复杂的指令，如定点乘除法指令、浮点指令以及访存指令，在执行阶段需要多拍。

1.1.1 取指和分支预测

龙芯 GS464 的流水线从取指流水级开始，每次取四条指令，但每次取指不能跨越 32 字节的指令 Cache 行。龙芯 GS464 取指时同时访问指令 Cache 和指令 TLB（简称 ITLB）。为了降低延迟，比较在取指阶段进行，Tag 但根据 Tag 比较结果进行指令选择在预译码阶段进行。取指过程中发生指令 Cache 不命中时向二级 Cache 发出访问请求。

16 项的 ITLB 是主 TLB 的子集。当 ITLB 不命中时，龙芯 GS464 产生一个内部指令去查找主 TLB 并且填充 ITLB，如果在主 TLB 中也不命中将产生一个普通的 TLB 例外。

取指后的流水级是预译码流水级。这一级的主要工作是预测转移指令的跳转方向以及目标地址。不同的转移指令使用不同的方式进行预测：Likely 类转移指令和直接跳转指令总是被预测为跳转，编译器可以通过编译出 Likely 指令进行静态预测；条件转移指令通过 BHT 预测跳转方向；间接跳转指令则用转移目标表（BTB）或返回地址栈（RAS）预测目标地址。

BHT 包括一个 9 位的全局历史寄存器（GHR）和一个 2K 项的模式历史表（PHT）。PHT 的每项是一个两位的饱和计数器，预测正确时计数器加 1，预测错误时计数器减 1。当计数器的值大于等于 2 时预测跳转成功。

16 项的 BTB 用于预测寄存器跳转指令的目标地址。每项 BTB 保存转移指令的地址和目标地址，以及一个两位的饱和计数器。当发生替换时，计数器的值小于 2 的项会优先被替换。

MIPS 指令集中没有 Call 和 Return 指令，通常使用转移链接（Jump and Link）指令和 jr31 指令进行函数调用和返回。龙芯 GS464 实现 4 项的返回地址栈 RAS。当译码出转移链接指令时将它的 PC + 8 压入 RAS，当译码出 jr31 指令时弹出 RAS 的顶作为 jr31 的目标地址。

龙芯 GS464 的第三级流水级是译码流水级。在这一级，四条指令被译成龙芯 GS464 的内部指令格式送往寄存器重命名模块。由于定点乘法指令和定点除法指令要生成两个 64 位结果，所以被译为两条内部指令。为了简化转移指令的管理，龙芯 GS464 每拍最多只进行一条转移指令的译码。

1.1.2 寄存器重命名

龙芯 GS464 使用物理寄存器堆的方法进行寄存器重命名，其中定点和浮点物理寄存器堆各为 64 项。龙芯 GS464 通过两个 64 项的物理寄存器映射表（Physical Register Mapping Table, 简称 PRMT）来保存物理寄存器和结构寄存器间的映射关系。

龙芯 GS464 的每个物理寄存器都处于以下四个状态的其中一个：MAP_EMPTY 表示该物理寄存器没有使用，MAP_MAPPED 表示该物理寄存器已经被映射但相应的值没有写回，MAP_WTBK 表示该物理寄存器的值已经写回，MAP_COMMIT 表示该物理寄存器值已经确定为处理器状态。

在寄存器重命名流水级，每条指令通过查找 PRMT 表得到该指令的两个源寄存器 SRC1、SRC2，和一个目标寄存器 DEST 当前所对应的物理寄存器号 PSRC1，PSRC2 和 ODEST。同时为目标寄存器 DEST 分配一个状态为 MAP_EMPTY 的一个物理寄存器 PDEST，新分配的物理寄存器的状态改为 MAP_MAPPED。同时修改 PRMT 表示 PDEST 是结构寄存器 DEST 的最新的映射。

在查找 PRMT 表建立逻辑寄存器和物理寄存器之间映射关系的同时，还需要检查同一拍重命名的四条指令间的相关性。如果某条指令 A 的源寄存器 SRC1 和同一拍前面指令 B 的目的寄存器 DEST 相同，A 的 SRC1 对应的物理寄存器改为 B 新分配的 PDEST，则而非 A 从 PRMT 中查出的 PSRC1。相同的原则也适用 PSRC2 和 ODEST。

经过寄存器重命名，物理寄存器号 PSRC1，PSRC2 和 PDEST 替换了原来指令中的结构寄存器号 SRC1，SRC2 和 DEST。其中物理寄存器号 PSRC1 和 PSRC2 送到保留站用于判断指令间的数据相关；ODEST 域保存在 ROQ 中，在指令提交时用于释放物理寄存器。

指令执行时，该指令的 PDEST 对应的 PRMT 的项设为 MAP_WTBK，表示该寄存器的值已经准备好了，后面的指令可以使用该寄存器的值。

指令提交时，该指令的 PDEST 对应的 PRMT 项设为 MAP_COMMIT 状态，ODEST 对应的 PRMT 项设为 MAP_EMPTY 状态，表示为该指令新分配的目的寄存器 PDEST 成为处理器状态，并释放该指令的目标寄存器原来所对应的物理寄存器。

从上面的寄存器重命名的过程可以看出一个结构寄存器可能同时对应多个物理寄存器，即一个逻辑寄存器在流水线中由于被多条指令修改可能有一系列的值。与一个结构寄存器对应的多个物理寄存器除了有一个表示该结构寄存器的处理器状态外，其它的分别对应于在流水线中的写该逻辑寄存器的多条指令。每个物理寄存器在每次分配之后只会被写一次。

1.1.3 指令发射和读寄存器

寄存器重命名后的指令送到保留站调度执行。龙芯 GS464 具有两个独立的分组保留站：定点和访存指令送到定点保留站；浮点指令送到浮点保留站。每一个保留站 16 项。

在寄存器重命名阶段，每条指令查找 PRMT 表确定操作数是否在寄存器堆中。如果查找 PRMT 表时相应的操作数没有准备好，该指令在送入保留站的途中以及在保留站中要通过比较自己的源寄存器号和结果总线或 Forward 总线的目标寄存器号以确定源操作数何时准备好。结果总线和 Forward 总线来自五个功能部件，结果总线送出指令的执行结果以及目标寄存器号，而 Forward 总线预测下一拍会被送出的结果以及相应指令的目标寄存器号。

两个保留站每拍最多可以发射五个源操作数准备好的指令到五个功能部件。如果在保留站中同一个功能部件有多个操作数准备好的指令，则选择最“老”的指令进行发射。在保留站中用一个 AGE 域来记录每一条指令在保留站中的“年龄”。

从保留站发射的指令到寄存器堆中读操作数后再送到功能部件执行。龙芯 GS464 有一个定点寄存器堆和一个浮点寄存器堆，大小都是 64*64。定点寄存器堆有 3 个写端口和 7 个读端口，其中 ALU1 使用 1 个写端口和 3 个读端口，ALU2 和访存部件各使用 1 个写端口和 2 个读端口。浮点寄存器堆有 3 个写端口和 7 个读端口，其中两个浮点部件各使用 1 个写端口和 3 个读端口，访存部件使用 1 个写端口和 1 个读端口用于浮点取数和存数指令。定点和浮点寄存器间的数据传输指令，如 MTC1、DMTC1、MFC1、DMFC1、CTC1 和 CFC1 使用访存数据通路传输数据，因此由访存部件执行。

特殊指令如 Branch and Link 指令的程序计数器或条件转移指令的 Taken 位从转移队列中读出并且与寄存器堆中的操作数一起送到相应的功能部件。

1.1.4 指令执行和功能部件

指令从寄存器堆中读取操作数后根据指令的类型送到相应的功能部件或访存部件执行，龙芯 GS464 包括两个定点部件 ALU1 和 ALU2，两个浮点部件 FALU1 和 FALU2。

定点 ALU1 执行定点加减、逻辑运算、移位、比较、Trap、以及转移指令。所有 ALU1 执行的指令 1 拍完成执行并写回。

定点 ALU2 执行定点加减、逻辑运算、移位、比较、以及乘除指令。定点乘法为全流水操作，延迟为 4 拍；定点除法采用 SRT 算法，非全流水操作，延迟根据操作数的不同从 4 拍到 37 拍不等；所有其他 ALU2 执行的指令 1 拍完成执行并写回。

浮点 FALU1 执行浮点加减、浮点乘法、浮点乘加（减）、取绝对值、取反、精度转换、定浮点格式转换、比较、转移等指令。FALU1 的所有运算为全流水操作。其中浮点取绝对值、取反、精度转换、比较、转移延迟为 2 拍，定浮点间格式转换延迟为 4 拍，浮点加减、浮点乘法、浮点乘加（减）延迟为 6 拍。

浮点 FALU2 执行浮点加减、浮点乘法、浮点乘加（减）、浮点除法、浮点开平方操作。其中浮点加减、浮点乘法、浮点乘加（减）为全流水操作，延迟为 6 拍；浮点除法和浮点开平方使用 SRT 算法，为非全流水操作，根据操作数的不同，单/双精度浮点除法延时从 4 到 10/17 拍不等，单/双精度开方延时从 4 到 16/31 拍不等。

除了执行 MIPS III 浮点指令外，浮点功能部件还可以执行并行单精度浮点指令，即在 64 位数据通路上同时计算两个单精度操作（加、减、乘和乘加）。另外，浮点功能部件还通过扩展浮点指令的格式域（FMT）执行 8/16/32/64 位 SIMD 多媒体定点指令。

1.1.5 指令提交和 Reorder 队列

在龙芯 GS464 中，指令顺序译码和重命名，乱序发射和执行，但有序结束。Reorder 队列 (Reorder Queue，简称 ROQ) 负责指令的有序结束，它按照程序次序保存流水线中所有已经完成寄存器重命名但未提交的指令。指令执行完并写回后，ROQ 按照程序次序提交这些指令。ROQ 最多可以同时容纳 64 条指令。

每条完成寄存器重命名的指令在送入保留站的同时也送入 ROQ。新进入的指令置为 ROQ_MAPPED 状态。指令写回后，ROQ 中的普通指令置为 ROQ_WTBK，转移指令置为 ROQ_BRWTBK 状态。状态为 ROQ_BRWTBK 的转移指令通过转移总线送到处理器的其他部分根据转移指令的执行结果修正转移猜测表并在转移猜测错误的情况下取消转移指令及其后续指令，并把状态置为 ROQ_WTBK。ROQ_WTBK 状态的指令在成为 ROQ 的队列头时可以提交。

ROQ 一拍最多可以提交队列头上的四条 ROQ_WTBK 状态指令。提交指令的 PDEST 和 ODEST 域送到寄存器重命名模块确认 PDEST 项的重命名为处理器状态并释放 ODEST 项的映射，它还通知访存队列相应的 Store 指令可以开始修改存储器。

为了实现精确例外，在指令执行过程中发生例外时把例外原因记录在 ROQ 相应的项中。当例外指令成为 ROQ 的队列头时进行例外处理，把例外原因、例外指令的 PC 值等例外信息记录到有关的 CP0 寄存器中，并根据例外类型把例外处理程序的入口地址送到程序计数器 PC 中。

1.1.6 移取消和转移队列

转移指令在重命名后进入 ROQ 和保留站的同时进入转移队列。转移队列同时可以容纳多达 8 条转移指令。

当转移指令发射执行时，转移队列提供该指令执行所需的信息，这些信息包括转移指令的 PC 值，和条件转移指令的预测 Taken 位等。

转移指令执行后，结果写回到转移队列。这些结果包括 JR 和 JALR 指令的目标地址、条件转移指令的转移方向和转移指令是否预测错误的标志位。转移指令的执行结果在提交前通过转移总线反馈到取指部分用来修正 BHT、BTB、RAS 和 GHR 以进行接下来的转移预测。

预测错误的指令和它后面的指令都需要取消。转移取消的一个核心问题是如何判断在流水线中乱序执行的指令哪些在取消的转移操作之前，哪些在取消的转移操作之后。龙芯 GS464 用转移指令把连续的指令流分为独立的基本块，并用转移指令在转移队列中的位置标识号 BRQID 对基本块进行编号。对于转移指令，这个标识表示它在转移队列中的位置；对于普通操作，这个标识表示它前面的转移指令在转移队列中的位置。通过这种方式，每一条指令都可以通过比较自己的 BRQID 和预测错误转移指令的 BRQID 确定它相对转移预测错误指令的位置。

1.1.7 存储访问与存储管理

龙芯 GS464 存储子系统对提高处理器的流水线效率起着重要作用。龙芯 GS464 一级指令和数据 Cache 大小均为 64KB，二级 Cache 大小为 512KB，均采用四路组相联结构；龙芯 GS464 片内集成了 DDR 内存控制器接口；龙芯 GS464 的 TLB 共有 64 项，为全相联结构，每项映射一个奇数页和一个偶数页；龙芯 GS464 通过一个 24 项的访存队列和一个 8 项的失效队列来动态解决存储相关，实现访存指令乱序执行、非阻塞 Cache、Load 猜测执行和写合并等。

龙芯 GS464 访存流水线分为 4 级。发射流水级把访存请求发射到地址运算部件后，第一拍通过地址运算部件计算虚地址并把访存请求送到 TLB 和 Cache；第二拍在 TLB 把虚地址转换为物

理地址的同时访问 Cache；第三拍根据 TLB 和 Cache 的访问结果确定 Cache 是否命中并送到访存队列；第四拍把访问结果写回。

龙芯 GS464 的存储系统使用 40 位虚地址和 40 位物理地址，并通过一个全相联的 TLB 进行虚实地址转换。TLB 包含一个 CAM 部分进行虚地址的全相联查找以及一个 RAM 该部分存储物理页号和页的保护位。龙芯 GS464 的 TLB 有 64 项，采用全相联结构，每项可以映射一个奇页和一个偶页。龙芯 GS464 的 TLB 的一个重要特点是它的页执行保护功能，它是通过在 TLB 的每一项增加一个执行保护位来实现的。该位可以由软件进行设置，表示相应的页是否可以被执行。硬件在取指过程中访问 TLB 时，除了做常规的权限检查外还进行可执行检查，如果取指时相应的页被置为不可执行，就会发生地址错例外。在操作系统中，只要利用上述方法对堆栈所在地址空间进行取指保护，就可以有效防范大多数利用缓冲区溢出技术进行的非法攻击。

龙芯 GS464 的一级数据 Cache 有 64KB，为四路组相联结构，块大小为 32 字节，采用随机替换算法。该 Cache 采用虚地址 Index 以及物理地址 Tag 以进行并行的 Cache 和 TLB 查找。由于 Cache 的每一路包含 16K 字节（是最小虚页的 4 倍），虚地址 Index 的两位（13:12）可能与物理地址 Tag 的相应位不相等，操作系统需要通过页着色（Page Coloring）或增加页的大小（每页 16KB 以上）来解决虚地址 Index 引起的不一致问题。龙芯 GS464 Cache 的数据和标志部分都采用单端口 RAM。为了降低 Cache 访问冲突，龙芯 GS464 把大小为 512*256 位的的每一路 Cache 分为四个 512*64 位的体，并允许对不同体的读和写同时进行以降低 Cache 访问冲突。在 Cache 失效时的回填操作 Refill、地址运算后访问 Cache、以及存数操作提交后的写回三种操作访问 Cache 端口冲突时，Refill 具有最高的优先级，存数操作的写回具有最低的优先级。

访存队列是龙芯 GS464 存储子系统的核心部件。它记录最多 24 个未执行完的 Load 或 Store 操作。虽然 Load 和 Store 操作乱序进入队列，但在访存队列中按它们程序中出现的次序排列。访存队列允许 Cache 失效的访存操作后面的多个 Cache 失效或命中的访存操作继续进行。龙芯 GS464 在 Cache 失效或访存相关时不重新进行访存，访存队列通过物理地址的全相联比较动态解决访存操作间的相关。取数操作进入队列时，通过地址比较按字节接收它前面的最近一个对同一地址的存数操作的值；存数操作进入队列时，通过地址比较按字节把所存的值传递给它后面对同一地址的取数操作，直到下一个对同一地址的存数操作。

一级 Cache 失效的取指或访存操作或被送入失效队列（Miss Queue）。失效队列处于指令 Cache、数据 Cache、二级 Cache、DDR 内存控制器接口、以及 SysAD 系统总线接口之间。该队列接收一级 Cache 失效的取指或访存操作并访问二级 Cache，并把相应的访问结果通过回填总线送回一级 Cache；在二级 Cache 访问失效时访问下一级存储器或系统总线接口，并把相应的访问结果送回二级以及一级 Cache。龙芯 GS464 的失效队列还支持失效写合并（Store Fill Buffer）优化，即可以把多个对同一 Cache 块的写请求合并在一起组成完整的 Cache 块，避免了没必要的存储器访问。

龙芯 GS464 集成了片上二级 Cache，二级 Cache 的块大小为 32 字节，容量 512KB，采用四路组相联结构。龙芯 GS464 的 512KB 二级 Cache 由 64 个 1024*64 位的 RAM 块组成，每次访问二级 Cache 时，只要打开相应 RAM 块的片选使能以降低功耗。

龙芯 GS464 内部集成的内存控制器的设计遵守 DDR2 SDRAM 的行业标准（JESD79-2B），支持最大 4 个物理内存 Bank（由 4 个 DDR2 SDRAM 片选信号实现），一共含有 15 位的地址总线（13 位行列地址总线和 2 位逻辑 Bank 总线）。龙芯 GS464 内部集成的内存控制器实现了一种动态的 Page 管理策略，针对一次访存操作，内存控制器对 Open Page 策略/Close Page 策略的选择是由硬件电路来实现的，无需软件设计人员来干预。

1.2 Misc

FIXME: merge this into the intro chapter...龙芯 GS464 处理器核只支持一种地址模式，一种指令集模式和一种尾端模式龙芯 GS464 处理器核只支持 64 位的虚拟地址模式，并且硬件保证兼容 32 位的地址模式。龙芯 GS464 处理器核实现了完备的 MIPS64 R2 指令集，另外还增加了一些整型和浮点指令。龙芯 GS464 处理器核只工作在小尾端模式。

第二章　指令集概述

龙芯 GS464 处理器核完全兼容 MIPS64 R2 体系结构，提供了其定义的全套指令（尽管龙芯 GS464 处理器核对所有 MIPS64 R2 指令都作了支持，但在实现上对一些的指令作了重定义。具体的细节请参照 MIPS 兼容性一章 9.1.1 节）。在此基础上，GS464 核还提供了（1）整点、浮点以及访存扩展指令：它们有助于提高常见应用的性能；（2）多媒体（SIMD）指令：有助于提高对越来越多的多媒体应用的支持；（3）x86 虚拟机扩展指令，用于在硬件层面对 x86 虚拟机的支持。

本章主要列出了 GS464 处理核的所有处理器核和 CP0 指令。其他指令，如浮点指令将在第七章，多媒体指令则在附录C中具体介绍。

2.1　指令编码

龙芯处理器指令编码为 32 位，主要包括三种指令格式：立即数型（I-型），跳转型（J-型）和寄存器型（R-型）指令格式。这三种指令的格式分别表示在图 2.1中。

图 2.1: 龙芯处理器指令格式

立即数型（I-Type）：

31　26	25　21	20　16	15　0
op	rs	rs	immediate

跳转型（J-Type）：

31　26	25　0
op	target

寄存器型（R-Type）：

31　26	25　21	20　16	15　11	10　6	5　0
op	rs	rt	rd	sa	funct

其中各位域解释为：

op:	6 位操作码;	target:	26 位跳转目标地址;
rs:	5 位源操作寄存器域;	rd:	5 位目的操作寄存器域;
rt:	5 位目标操作寄存器;	sa:	5 位移位位数;
immediate:	16 位立即数;	funct:	6 位功能域。

2.2 MIPS64 兼容指令列表

根据指令功能，MIPS64 指令可分为如下几组：访存指令、运算指令、转移指令、其它指令、和协处理器指令等。下面小节分组列出龙芯 GS464 处理器核兼容的所有 MIPS64 指令。

2.2.1 访存指令

MIPS 体系结构采用 Load/Store 架构。所有运算都在寄存器上进行，只有访存指令可对主存中的数据进行访问。访存指令包含各种宽度数据的读写、无符号读、非对齐访存和原子访存等。

表 2.1: MIPS64 访存指令

指令 OpCode	描述	MIPS ISA
LB	取字节	MIPS32
LBU	取无符号字节	MIPS32
LH	取半字	MIPS32
LHU	取无符号半字	MIPS32
LW	取字	MIPS32
LWU	取无符号字	MIPS32
LWL	取字左部	MIPS32
LWR	取字右部	MIPS32
LD	取双字	MIPS64
LDL	取双字左部	MIPS64
LDR	取双字右部	MIPS64
LL	取标志处地址	MIPS32
LLD	取标志处双字地址	MIPS64
SB	存字节	MIPS32
SH	存半字	MIPS32
SW	存字	MIPS32
SWL	存字左部	MIPS32
SWR	存字右部	MIPS32
SD	存双字	MIPS64
SDL	存双字左部	MIPS64
SDR	存双字右部	MIPS64
SC	满足条件下存	MIPS32
SCD	满足条件下存双字	MIPS64

2.2.2 运算指令

运算指令用于完成寄存器值的算术、逻辑、移位、乘法和除法等操作。运算型指令有两种指令格式：寄存器指令格式（R-型，操作数和运算结果均保存在寄存器中）和立即数指令格式（I-型，其中一个操作数为一个 16 位的立即数）。

表 2.2: 运算指令

指令 OpCode	描述	MIPS ISA
算术指令（ALU 立即数）		
ADDI	加立即数	MIPS32
DADDI	加双字立即数	MIPS64
ADDIU	加无符号立即数	MIPS32
DADDIU	加无符号双字立即数	MIPS64
SLTI	小于立即数设置	MIPS32
SLTIU	无符号小于立即数设置	MIPS32
ANDI	与立即数	MIPS32
ORI	或立即数	MIPS32
XORI	异或立即数	MIPS32
LUI	取立即数到高位	MIPS32
算术指令（3 操作数）		
ADD	加	MIPS32
DADD	双字加	MIPS64
ADDU	无符号加	MIPS32
DADDU	无符号双字加	MIPS64
SUB	减	MIPS32
DSUB	双字减	MIPS64
SUBU	无符号减	MIPS32
DSUBU	无符号双字减	MIPS64
SLT	小于设置	MIPS32
SLTU	无符号小于设置	MIPS32
AND	与	MIPS32
OR	或	MIPS32
XOR	异或	MIPS32
NOR	或非	MIPS32
算术指令（2 操作数）		
CLO	字前导 1 个数	MIPS32
DCLO	双字前导 1 个数	MIPS64
CLZ	字前导 0 个数	MIPS32
DCLZ	双字前导 0 个数	MIPS64
WSBH	半字字节交换	MIPS32 R2
DSHD	字半字交换	MIPS64 R2
SEB	字节符号扩展	MIPS32 R2
SEH	半字符号扩展	MIPS32 R2
INS	位插入	MIPS32 R2
EXT	位提取	MIPS32 R2
DINS	双字位插入	MIPS64 R2

未完待续

表 2.2: 运算指令（续）

指令 OpCode	描述	MIPS ISA
DINSM	双字位插入	MIPS64 R2
DINSU	双字位插入	MIPS64 R2
DEXT	双字位提取	MIPS64 R2
DEXTM	双字位提取	MIPS64 R2
DEXTU	双字位提取	MIPS64 R2
整数乘除指令		
MUL	乘到通用寄存器	MIPS32
MULT	乘	MIPS32
DMULT	双字乘	MIPS64
MULTU	无符号乘	MIPS32
DMULTU	无符号双字乘	MIPS64
MADD	乘加	MIPS32
MADDU	无符号乘加	MIPS32
MSUB	乘减	MIPS32
MSUBU	无符号乘减	MIPS32
DIV	除	MIPS32
DDIV	双字除	MIPS64
DIVU	无符号除	MIPS32
DDIVU	无符号双字除	MIPS64
MFHI	从 hi 寄存器取数到通用寄存器	MIPS32
MTHI	从通用寄存器存数到 hi 寄存器	MIPS32
MFLO	从 lo 寄存器取数到通用寄存器	MIPS32
MTLO	从通用寄存器存数到 lo 寄存器	MIPS32
移位指令		
SLL	逻辑左移	MIPS32
SRL	逻辑右移	MIPS32
SRA	算术右移	MIPS32
SLLV	可变的逻辑左移	MIPS32
SRLV	可变的逻辑右移	MIPS32
SRAV	可变的算术右移	MIPS32
ROTR	循环右移	MIPS32 R2
ROTRV	可变的循环右移	MIPS32 R2
DSLL	双字逻辑左移	MIPS64
DSRL	双字逻辑右移	MIPS64
DSRA	双字算术右移	MIPS64
DSLLV	可变的双字逻辑左移	MIPS64
DSRLV	可变的双字逻辑右移	MIPS64
DSRAV	可变的双字算术右移	MIPS64

未完待续

表 2.2: 运算指令（续）

指令 OpCode	描述	MIPS ISA
DSLL32	双字逻辑左移 +32	MIPS64
DSRL32	双字逻辑右移 +32	MIPS64
DSRA32	双字算术右移 +32	MIPS64
DROTR	双字循环右移	MIPS64 R2
DROTR32	双字循环右移 +32	MIPS64 R2
DROTRV	双字可变的循环右移	MIPS64 R2

2.2.3 跳转分支指令

跳转和分支指令可改变程序的控制流，包括以下四种类型：

- PC 相对条件分支
- PC 无条件跳转
- 寄存器绝对跳转
- 过程调用

MIPS 定义中，所有转移指令后都紧跟一条延迟槽指令。likely 转移指令的延迟槽只在转移成功时执行，unikely 转移指令延迟槽指令总会得到执行。过程调用指令的返回地址默认保存在第 31 号寄存器中，根据第 31 号寄存器跳转将被认为从被调用过程返回。

表 2.3: 跳转分支指令

指令 OpCode	描述	MIPS ISA
J	跳转	MIPS32
JAL	立即数调用过程	MIPS32
JR	跳转到寄存器指向的指令	MIPS32
JR.HB	跳转到寄存器指向的指令	MIPS32 R2
JALR	寄存器调用过程	MIPS32
JALR.HB	寄存器调用过程	MIPS32 R2
BEQ	相等则跳转	MIPS32
BNE	不等则跳转	MIPS32
BLEZ	小于等于 0 跳转	MIPS32
BGTZ	大于 0 跳转	MIPS32
BLTZ	小于 0 跳转	MIPS32
BGEZ	大于或等于 0 跳转	MIPS32
BLTZAL	小于 0 调用过程	MIPS32
BGEZAL	大于或等于 0 调用过程	MIPS32
BEQL	相等则 Likely 跳转	MIPS32
BNEL	不等则 Likely 跳转	MIPS32

未完待续

表 2.3: 跳转分支指令（续）

指令 OpCode	描述	MIPS ISA
BLEZL	小于或等于 0 则 Likely 跳转	MIPS32
BGTZL	大于 0 则 Likely 跳转	MIPS32
BLTZL	小于 0 则 Likely 跳转	MIPS32
BGEZL	大于或等于 0 则 Likely 跳转	MIPS32
BLTZALL	小于 0 则 Likely 调用过程	MIPS32
BGEZALL	大于或等于 0 则 Likely 调用过程	MIPS32

2.2.4 协处理器指令

协处理器指令完成协处理器内部的操作。龙芯 GS464 处理器核有两个协处理器：0 号协处理器（CP0, 系统处理器）和 1 号协处理器（FPU, 浮点协处理器）。

- 0 号协处理器（CP0）通过 CP0 的寄存器来管理内存和处理异常。这些指令列在表 2-8 中。
- 1 号协处理器（CP1）指令包括浮点指令，多媒体指令，和龙芯扩展的定点计算指令。这些指令都是在浮点寄存器上操作。第 7 章将会对这些协处理器 1 指令进行总结。

龙芯处理器能够处理硬件中的流水线相关，包括 CP0 相关和访存相关，因此 CP0 指令并不需要 NOP 指令来校正指令序列。

表 2.4: CP0 指令

指令 OpCode	描述	MIPS ISA
DMFC0	从 CP0 寄存器取双字	MIPS32
DMTC0	往 CP0 寄存器写双字	MIPS32
MFC0	从 CP0 寄存器取	MIPS32
MTC0	往 CP0 寄存器写	MIPS32
TLBR	读索引的 TLB 项	MIPS32
TLBWI	写索引的 TLB 项	MIPS32
TLBWR	写随机的 TLB 项	MIPS32
TLBP	在 TLB 中搜索匹配项	MIPS32
CACHE	Cache 操作	MIPS32
ERET	异常返回	MIPS32
DI	禁止中断	MIPS32 R2
EI	允许中断	MIPS32 R2
DERET	Debug 返回	EJTAG
RDHWR	读取硬件寄存器	MIPS32 R2
RDPGPR	从影子寄存器中读取	MIPS32 R2
WRPGPR	写到影子寄存器	MIPS32 R2
SDBBP	软件断点	EJTAG

2.2.5 其它指令

MIPS64 中，除了前面列出上述指令外还有其它一些指令，详见表 2-9 至表 2-12：

表 2.5: MIPS64 其他指令

指令 OpCode	描述	MIPS ISA
特殊指令		
SYSCALL	系统调用	MIPS32
BREAK	断点	MIPS32
SYNC	同步	MIPS32
SYNCI	同步指令缓存	MIPS32 R2
异常指令		
TGE	大于或等于陷入	MIPS32
TGEU	无符号数大于或等于陷入	MIPS32
TLT	小于陷入	MIPS32
TLTU	无符号数小于陷入	MIPS32
TEQ	等于陷入	MIPS32
TNE	不等陷入	MIPS32
TGEI	大于或等于立即数陷入	MIPS32
TGEIU	大于或等于无符号立即数陷入	MIPS32
TLTI	小于立即数陷入	MIPS32
TLTIU	小于无符号立即数陷入	MIPS32
TEQI	等于立即数陷入	MIPS32
TNEI	不等于立即数陷入	MIPS32
条件移动指令		
MOVF	条件移动当浮点条件假	MIPS32
MOVN	条件移动当通用寄存器非 0	MIPS32
MOVT	条件移动当浮点条件真	MIPS32
MOVZ	条件移动当通用寄存器为 0	MIPS32
空操作指令		
PREF	预取指令	MIPS32
PREFX	预取指令	MIPS32
NOP	空操作	MIPS32
SSNOP	单发射空操作	MIPS32

2.3 扩展指令

龙芯 GS464 处理器在 MIPS64 指令集的基础上为提高某些应用的性能对指令集有所扩展，主要包括：

- 带偏移访存指令；

- 四字访存指令;
- 单目标定点乘除指令。

表 2.6: 龙芯扩展指令

指令 OpCode	描述	MIPS ISA
访存指令		
gsLDX	带偏移取双字	GS464
gsLDXC1	带偏移取双字	GS464
gsLWX	带偏移取字	GS464
gsLWXC1	带偏移取字	GS464
gsSDX	带偏移存双字	GS464
gsSDXC1	带偏移存双字	GS464
gsSWX	带偏移存字	GS464
gsSWXC1	带偏移存字	GS464
gsLQ	取四字	GS464
gsSQ	存四字	GS464
整数乘除指令		
MULT.G	龙芯乘	GODSON2
DMULT.G	龙芯双字乘	GODSON2
MULTU.G	龙芯无符号乘	GODSON2
DMULTU.G	龙芯无符号双字乘	GODSON2
DIV.G	龙芯除	GODSON2
DDIV.G	龙芯双字除	GODSON2
DIVU.G	龙芯无符号除	GODSON2
DDIVU.G	龙芯无符号双字除	GODSON2
MOD.G	龙芯求模	GODSON2
DMOD.G	龙芯双字求模	GODSON2
MODU.G	龙芯无符号求模	GODSON2
DMODU.G	龙芯无符号双字求模	GODSON2

第三章　CP0 控制寄存器

本章的主要内容是介绍零号协处理器各个寄存器及其位域含义。零号协处理器（Coprocessor 0，简称 CP0），作为计算机处理器重要的组成部分，是获取处理器的当前运行状态的主要信息来源，并可以用于控制及改变处理器的状态。表 3.1 列出了龙芯 GS464 支持的所有 CP0 寄存器。

表 3.1: CP0 寄存器表

寄存器号		寄存器名	位宽	描述
总号	子号			
0	0	Index	32	TLB 表项索引寄存器
1	0	Random	32	TLB 表项伪随机计数寄存器
2	0	EntryLo0	64	TLB 表项低半部分偶虚页入口寄存器
3	0	EntryLo1	64	TLB 表项低半部分奇虚页入口寄存器
4	0	Context	64	页表入口（PTE）指针寄存器（32 位寻址模式）
5	0	PageMask	32	页面掩码寄存器
5	1	PageGrain	32	页面颗粒度寄存器
6	0	Wired	32	固定连线寄存器（不用随机替换的低端 TLB 表项数）
7	0	HWREna	32	硬件寄存器使能寄存器
8	0	BadVAddr	64	错误虚地址寄存器
9	0	Count	32	计数寄存器
10	0	EntryHi	64	TLB 表项高半部分入口寄存器（虚页号和 ASID）
11	0	Compare	32	比较寄存器
12	0	Status	32	状态寄存器
12	1	IntCtl	32	向量中断控制寄存器
12	2	SRSCtl	32	影子寄存器组控制寄存器
13	0	Cause	32	例外原因寄存器
14	0	EPC	64	例外程序计数寄存器
15	0	PRid	32	处理器版本标识寄存器
15	1	EBase	32	例外向量基地址寄存器
16	0-3	Config	32	配置寄存器，和辅助配置寄存器 1、2、3
17	0	LLAddr	64	链接加载地址寄存器
18	—	—	—	保留
19	—	—	—	保留
20	0	Xcontext	64	页表入口（PTE）指针寄存器（64 位寻址模式）
21	—	—	—	保留

未完待续

表 3.1: CP0 寄存器表（续）

寄存器号		寄存器名	位宽	描述
总号	子号			
22	0	Diagnostic	32	诊断寄存器：使能/禁用 BTB，RAS 以及清空 ITLB
23	0	Debug	64	EJTAG 调试寄存器
24	0	DEPC	64	EJTAG 调试例外程序计数器
25	0,2	PerfCtl	32	性能控制寄存器 0，1
25	1,3	PerfCnt	64	性能计数寄存器 0，1
26	0	ErrCtl	64	Parity/ECC 校验控制寄存器
27	0	CacheErr	64	Cache 校验错误寄存器
27	1	CacheErr1	64	Cache 校验错误虚地址寄存器
28	0	TagLo	32	Cache TAG 低半寄存器
28	1	DataLo	64	数据低半寄存器：Cache 数据队列交互和诊断
29	0	TagHi	32	Cache TAG 高半寄存器
29	1	DataHi	64	数据高半寄存器：Cache 数据队列交互和诊断
30	0	ErrorEPC	64	错误例外程序计数寄存器
31	0	DESAVE	64	调试暂存寄存器器

这些 CP0 寄存器的值可以通过 MFC0/DMFC0 指令来获取，或者通过 MTC0/DMTC0 指令来设置。表 2.4 列出了所有的龙芯 GS464 处理器核 CP0 指令。注意，只有当处理器运行在核心模式时或状态寄存器（Status 寄存器）中的第 28 位（CU[0]）被设置时，才可以使用 CP0 指令，否则将产生一个“协处理器不可用例外”。

3.1 CP0 寄存器详解

下面各小节将对每个寄存器进行详细说明。以下对寄存器位域的注解中，如果一个位域为保留，则该位域必须写入其保留值（一般为 0），并在读的时返回保留值：在保留字段写入非保留值的结果为**未定义**；如果一个域为只读，则在读的时候，返回其保留值，而任何写入值将被忽略。

龙芯 GS464 实现了对 48 位物理地址的支持（见 PageGrain 寄存器3.1.6 节），并可以在 40 位和 48 位物理地址支持间切换。在以下各章节中，所有的描述主要基于 48 位物理地址模式，包括所有的示意图及 CP0 寄存器的位域解释。在3.2 节中，我们将简述 40 位和 48 位物理地址模式，可能影响到的寄存器位域。

3.1.1 Index 寄存器（0）

Index 寄存器是一个 32 位可读写寄存器。该寄存器的最高位表示最近的 TLB 探测 (TLBP) 指令执行是否成功，其最后六位为 Index 域，用于在 TLBR 和 TLBWI 指令中指示操作的 TLB 表项。表 3.2 给出了 Index 寄存器的格式，及各域的含义。

31	30 ... 6	5 ... 0
P	0	Index
1	25	6

位域	描述
P	探测失败位：最近一次 TLB 探测（TLBP）指令失败时置 1。
Index	TLB 表项索引域，用于指示 TLBR 和 TLBWI 指令操作。
0	保留：必须按 0 写入，读时返回 0。

表 3.2: CP0: Index 寄存器

3.1.2 Random 寄存器（1）

Random 寄存器是一个 32 位只读寄存器，其低六位为 Random 域，每执行完一条指令，该寄存器值减 1。它被用作伪随机数，用于指示 TLB 随机写指令操作的 TLB 项。Random 域的值在一个上下界之间浮动：

- 上界为 TLB 最大表项索引值，即总项数减 1，在 GS464 上其值为 63；
- 下界为 Wired 寄存器的值，即保留给操作系统专用的 TLB 表项数。

一般而言，无需读取 Random 寄存器值。不过该寄存器是可读的，主要用于验证处理器的操作是否正确。Random 寄存器在系统重起时将置为上界；此外，当 Wired 寄存器被重写时，该寄存器也会被自动置为上界。表 3.3 给出了 Random 寄存器的格式，及各域的含义。

31 ... 6	5 ... 0
0	Random
26	6

位域	描述
Random	伪随机 TLB 索引值域。
0	保留：必须按 0 写入，读时返回 0。

表 3.3: CP0: Random 寄存器

3.1.3 EntryLo0、EntryLo1 寄存器（2，3）

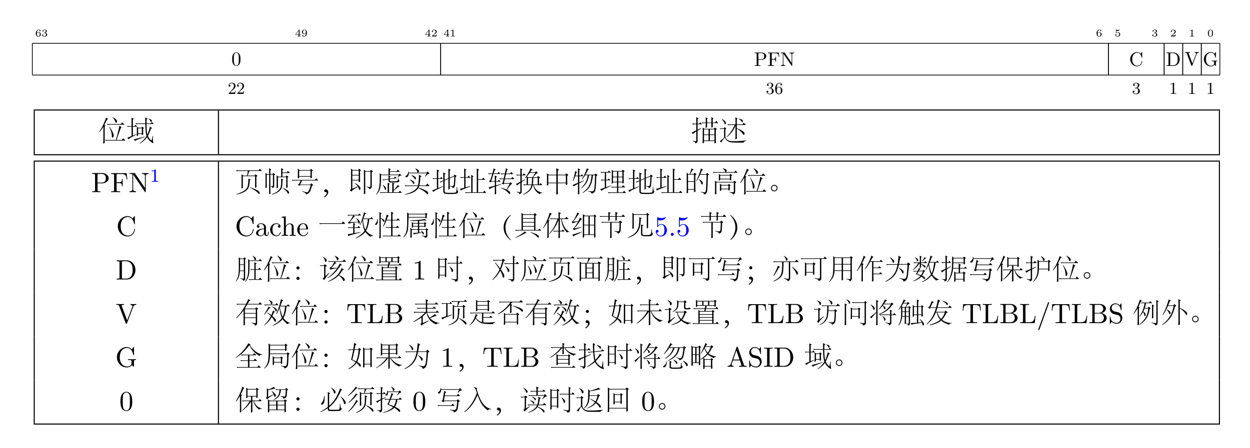
EntryLo 寄存器组包括两个具有相同格式的 64 位可读写寄存器：EntryLo0 和 EntryLo1。龙芯的 TLB 表项是按成对页面组织的，EntryLo0 和 EntryLo1 分别对应 TLB 表项的偶、奇虚页面。当执行 TLB 读写操作时，它们包含有当前 TLB 表项对应页面的页帧号（Page-frame number，简称 PFN）和其他页面信息。表 3.4 给出了 EntryLo 寄存器的格式，及各域的含义。

由于一个 TLB 表项对应两个页面，却只有一个全局位。所以在 TLB 写操作时，只有当两个页面全局位都为 1 时，TLB 的全局位才会被写入 1。关于 TLB 表项格式的具体信息见4.3.1 节。

3.1.4 Context 寄存器（4）

Context 寄存器是一个 64 位可读写寄存器，在 32 位地址模式下，包含一个指向页表（Pagetable entry，PTE）的指针。作为一个重要的操作系统数据结构，页表存储有虚拟地址到物理地址转换的信息。当 TLB 脱靶例外发生时，CPU 需要从页表中将缺失页的信息加载到 TLB 中。

[1]在 MIPS64 的文档中，该域被写作 PFNX 和 PFN 域。我们这里将其合并在一起，从概念和理解上都更直接。



63–42	41–6	5–3	2	1	0
0	PFN	C	D	V	G
22	36	3	1	1	1

位域	描述
PFN[1]	页帧号，即虚实地址转换中物理地址的高位。
C	Cache 一致性属性位（具体细节见5.5 节）。
D	脏位：该位置 1 时，对应页面脏，即可写；亦可用作为数据写保护位。
V	有效位：TLB 表项是否有效；如未设置，TLB 访问将触发 TLBL/TLBS 例外。
G	全局位：如果为 1，TLB 查找时将忽略 ASID 域。
0	保留：必须按 0 写入，读时返回 0。

表 3.4: CP0: EntryLo 寄存器

这时，操作系统将通过 Context 寄存器在页表中寻找当前缺失页的映射信息：这些信息一般存储在 KSEG0 段中。同时，Context 寄存器还包含有 BadVAddr 寄存器的部分信息，方便软件 TLB 例外的处理。表 3.5 给出了 Context 寄存器的格式，及各域的含义。

63–23	22–4	3–0
PTEBase	BadVPN2	0
41	19	4

位域	描述
PTEBase	页表入口数列基地址域。
BadVPN2	待转换虚地址虚页号域（VPN2）：由硬件在 TLB 例外时写入。
0	保留：必须按 0 写入，读时返回 0。

表 3.5: CP0: Context 寄存器

这里，在 32 位地址模式下，BadVPN2（即未能转换的虚页 VPN2）是一个 19 位的字段，它对应着待转换虚地址的 31:13 位：12 位被排除在外，是因为每个 TLB 表项对应着一个奇偶页面对。如果页面大小为 4K 字节，那么 VPN2 可以直接寻址按 8 个字节长的，按对组织的页表入口数列。对于更大尺寸的页面设置，则需要屏蔽掉 VPN2 中相应的多余位以产生合适的地址。

3.1.5 PageMask 寄存器（5）

PageMask 寄存器是一个 32 位可读写的寄存器：它包含了一个比较掩码域，用于设置了当前 TLB 表项对应的页面大小。表 3.6 给出了 PageMask 寄存器的格式，及 Mask 域的含义：未在表 3.6 中列出的 Mask 的值为无效的；寄存器的其他位保留，必须按 0 写入，读时返回 0。

进行虚实地址转换时，Mask 域的某位为 1 则表示虚地址的对应位将不用于地址比较，即页面地址在更高的位上对齐，也就是说，更大的页面。由上表可知，GS464 核支持的最小页面为 4K，最大为 16MB，每级之间以 4 的倍数增加。

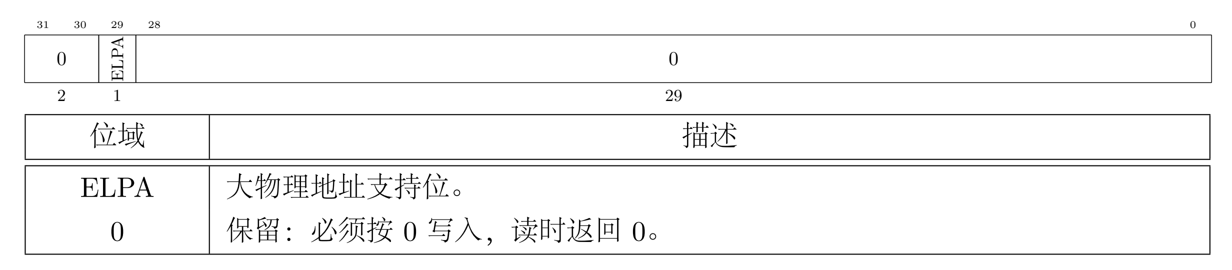
3.1.6 PageGrain 寄存器（5）

PageGrain 寄存器是一个 32 位可读写的寄存器，龙芯 GS464 只定义了这个寄存器的第 29 位：大物理地址支持位（Enable Large Physical Adress，简称 ELPA），其余位保留。当 ELPA=1 时，龙芯 GS464 支持 48 位物理地址；ELPA=0 时，GS464 只支持 40 位物理地址。表 3.7 给出了 PageGrain 寄存器的格式，及各域的含义。

31–25	24–13	12–0
0	Mask	0
7	12	13

页面大小	Mask 位											
	24	23	22	21	20	19	18	17	16	16	14	13
4KB	0	0	0	0	0	0	0	0	0	0	0	0
16KB	0	0	0	0	0	0	0	0	0	0	1	1
64KB	0	0	0	0	0	0	0	0	1	1	1	1
256KB	0	0	0	0	0	0	1	1	1	1	1	1
1MB	0	0	0	0	1	1	1	1	1	1	1	1
4MB	0	0	1	1	1	1	1	1	1	1	1	1
16MB	1	1	1	1	1	1	1	1	1	1	1	1

表 3.6: CP0: PageMask 寄存器



31–30	29	28–0
0	ELPA	0
2	1	29

位域	描述
ELPA	大物理地址支持位。
0	保留：必须按 0 写入，读时返回 0。

表 3.7: CP0: PageGrain 寄存器

在本手册中，所有的描述和示例都主要以 48 位物理地址模式为标准给出。关于不同物理地址模式的影响，请参见 3.2 节。

3.1.7 Wired 寄存器（6）

Wired 寄存器是一个 32 位可读写寄存器，该寄存器的值指定了 TLB 中固定连线表项与随机表项之间的界限。表 3.8 给出了 Wired 寄存器的格式，及各域含义。

31–6	5–0
0	Wired
26	6

位域	描述
Wired	TLB 固定表项大小域。
0	保留：必须按 0 写入，读时返回 0。

表 3.8: CP0: Wired 寄存器

如图 3.1 所示，Wired 表项是固定的不会被 TLB 随机写操作修改的 TLB 表项。Wired 寄存器在系统复位时置 0；写该寄存器的同时，Random 寄存器的值也会被置为其上限值（参阅前面 Random 寄存器的说明）。

3.1.8 HWREna 寄存器（7）

HWREna 寄存器是一个 32 位可读写寄存器，GS464 只定义了该寄存器的 Mask 域，用于表示在用户态下，指令 `RDHWR` 可以得到的硬件寄存器信息。当前，GS464 提供了四个硬件寄存器，

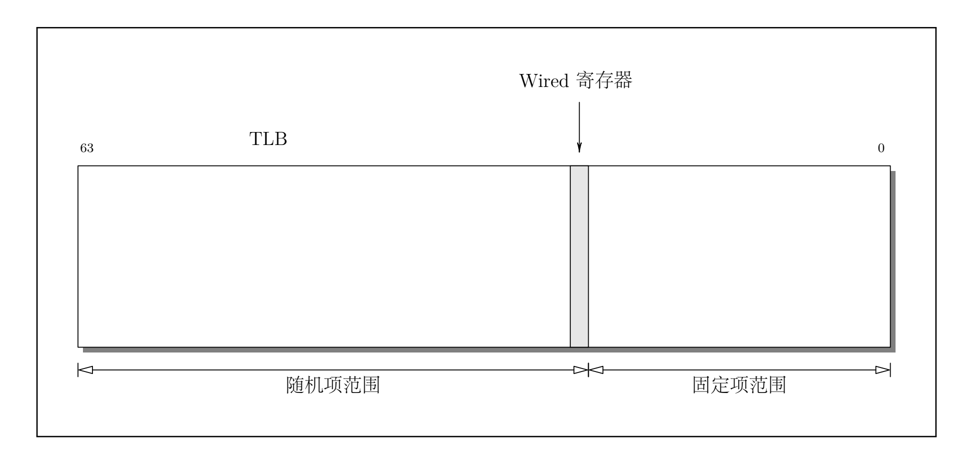


图 3.1: Wired 寄存器示意图

分别对应 Mask 域的四个位。表 3.9 给出了 HWREna 寄存器的格式，及 Mask 域对应的硬件寄存器。寄存器的其他位为保留，必须按 0 写入，读时返回 0。

31 4	3 0
0	Mask
28	4

Mask 位				硬件寄存器	描述
3	2	1	0		
0	0	0	1	CPUnum	运行当前程序的 CPU 数寄存器
0	0	1	0	SYNCI_Step	一级 Cache 行数目寄存器
0	1	0	0	CC	CP0 计数寄存器
1	0	0	0	CCRes	计数寄存器的精度

表 3.9: CP0: HWREna 寄存器

3.1.9 BadVAddr 寄存器（8）

BadVAddr 寄存器是一个 64 位只读寄存器。当软复位，NMI 或 Cache 错误例外发生时，BadVAddr 寄存器记录下导致 TLB 或寻址错误例外的虚拟地址，否则寄存器值将一直保持不变。在一般运行情况下，该寄存器的值为**未定义**。表 3.10 给出了 BadVAddr 寄存器的格式。

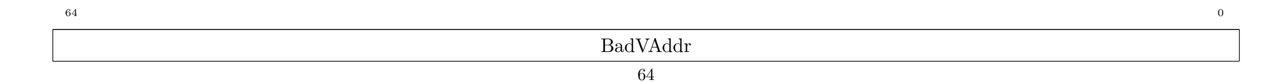


表 3.10: CP0: BadVAddr 寄存器

3.1.10 Count、Compare 寄存器（9，11）

Count 和 Compare 寄存器是一对 32 位可读写寄存器：Count 寄存器是一个实时的计数器，它的值每两个 CPU 时钟周期加 1；而 Compare 寄存器则用来根据 Count 寄存器的值在特定的时刻生成一个中断。当 Compare 寄存器被写入一个新值时，CPU 会不断地将其与 Count 寄存器中的值进行比较。一旦这两个值相等时，便会产生一个中断请求，同时将 Cause 寄存器的 TI 和

IP[7] 位置位。当 Compare 寄存器被再次重写时，Cause 寄存器的 TI 位会被清零。由于 GS464 核上实现了动态调频，Count 寄存器的计数频率不是固定的，其值总为当前 CPU 频率的一半。

Count 寄存器:

31–0
Count
32

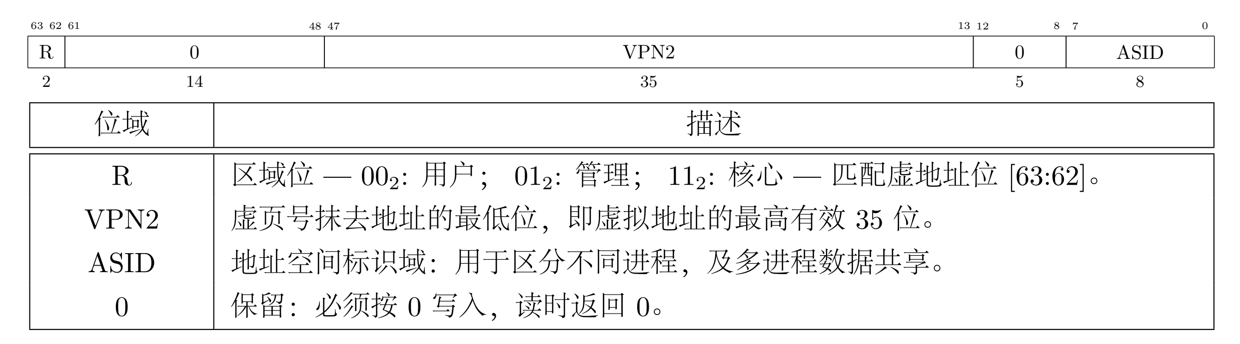
Compare 寄存器:

31–0
Compare
32

表 3.11: CP0: Count 和 Compare 寄存器

3.1.11 EntryHi（10）寄存器

EntryHi 寄存器是一个 64 位可读写寄存器，它用于在使用 TLB 指令时存放 TLB 表项的高位。表 3.10 给出了 EntryHi 寄存器的格式及各域的说明。注意，其实 VPN2 域是 64 位虚拟地址的 47:13 位，而 63:62 位为区域位。当 TLB 重填，TLB 无效，或者 TLB 修改例外发生时，引起例外的虚拟地址的虚拟页号（VPN2）和 ASID 将被加载到 EntryHi 寄存器中。



63–62	61–48	47–13	12–8	7–0
R	0	VPN2	0	ASID
2	14	35	5	8

位域	描述
R	区域位 — 00_2: 用户； 01_2: 管理； 11_2: 核心 — 匹配虚地址位 [63:62]。
VPN2	虚页号抹去地址的最低位，即虚拟地址的最高有效 35 位。
ASID	地址空间标识域：用于区分不同进程，及多进程数据共享。
0	保留：必须按 0 写入，读时返回 0。

表 3.12: CP0: EntryHi 寄存器

3.1.12 Status（12）寄存器

Status 寄存器是一个 32 位可读写寄存器，它包含有关于操作模式，中断允许和处理器状态诊断等丰富的信息。在龙芯 GS464 上，Status 寄存器复位时的初始值为 0x30C0_00E4。表 3.13 给出了 Status 寄存器的格式，及各域的含义。

表 3.13: CP0: Status 寄存器

31–28	27	26	25–24	23	22	21	20	19	18–16	15–8	7	6	5	4–3	2	1	0
CU[3:0]	0	FR	0	PX	BEV	0	SR	NMI	0	IM7-IM0	KX	SX	UX	KSU	ERL	EXL	IE
4	1	1	2	1	1	1	1	1	3	8	1	1	1	2	1	1	1

位域	描述
CU	协处理器使能域：某位为 1 则表明对应协处理器可用；初值是 0011_2。
FR	附加浮点寄存器使能位 — 0: 16 个浮点寄存器；1: 32 个浮点寄存器。
PX	用户模式下的 64 位操作使能位。其余模式下的 64 位操作无需使能。注意，此时用户模式下 64 位操作是否可用仍需要判断 UX 位。

未完待续

表 3.13: CP0: Status 寄存器（续）

位域	描述
BEV	启动向量（boot-entry vector）指示位 — 0: 正常运行；1：启动运行。
SR	软复位例外指示位。
NMI	NMI 例外指示位：注意，软件不能把这位由 0 写为 1。
IM	中断屏蔽域：如果某位被使能，则将允许对应中断触发。
KSU	运行模式位 — 11_1: 未定义； 10_2: 用户； 01_2: 管理； 00_2: 内核。
KX	64 位内核段访问使能位。
SX	64 位管理段访问使能位。
UX	64 位用户段访问使能位。
ERL	错误级指示位：复位，软复位，NMI 或 Cache 错误发生时，此位将置 1。
EXL	例外级指示位：复位，软复位或 Cache 错误以外的例外发生时，此位将置 1。
IE	中断使能位 — 0：禁用所有中断；1：使能所有中断。
0	保留：必须按 0 写入，读时返回 0。

一些关于 Status 寄存器重要字段的说明：

- 4 位的协处理器使能域，CU，控制着 4 个可能的协处理器的可用性。注意，不管 CU[0] 位如何设置，在内核模式下 CP0 总是可用的。
- 8 位的中断屏蔽域，IM，控制着 8 个中断的使能：一个中断必须被使能才可能被触发。同时，中断屏蔽域 IM 和 Cause 寄存器的中断待定域（IP）位一一对应，并合作处理中断发生的信息。更多细节请参考 Cause 寄存器的中断待定域。
- 中断的使能被多个条件控制。只有在以下条件符合时，IM 位中的设置才会生效：

  `IE == 1 && EXL == 0 && ERL ==  0`。

- 操作模式。龙芯 GS464 处理器有三种操作模式，他们分别是：用户模式，管理模式，和内核模式。由 Status 寄存器的值决定了处理器的工作模式，这些模式的设置列表如下。

KSU	ERL	EXL	运行模式
10_2	0	0	用户模式
01_2	0	0	管理模式
00_2	0	0	内核模式
	0	1	内核模式（例外状态）
	1		内核模式（错误状态）

表 3.14: CP0：处理器的工作模式

- 地址空间访问（具体空间划分见 4.2 节）。
  - 内核地址空间：当处理器处在内核模式时，可以访问内核地址空间；
  - 管理地址空间：当处理器处在内核或管理模式时，可以访问管理地址空间；
  - 用户地址空间：处理器在三种操作模式下都可以访问用户地址空间。

3.1.13 IntCtl（12/1）寄存器

龙芯 GS464 实现了 MIPS64 R2 体系中扩充的向量中断（vectored interrupt）支持。IntCtl 是一个 32 位可读写寄存器，它的 VS 域用来指示中断向量的向量空间。表 3.15 给出了 IntCtl 寄存器的格式，及 VS 域的编码与向量空间对应关系。寄存器的其他域：1 域为只读域；0 域为保留域，必须按 0 写入，读时返回 0。

31–26	25–10	9–5	4–0
1	0	VS	0
6	16	5	5

编码	向量空间（16 进制）	向量空间（10 进制）
0x00	0x000	0
0x01	0x020	32
0x02	0x040	64
0x04	0x080	128
0x08	0x100	256
0x10	0x200	512

表 3.15: CP0: IntCtl 寄存器

3.1.14 SRSCtl（12/2）寄存器

SRSCtl 寄存器是一个 32 位可读写寄存器，用于管理处理器的影子寄存器组。龙芯 GS464 中的 SRSCtl 寄存器只实现两个域：PSS 和 ESS。同时，因为龙芯 GS464 还没有实现影子寄存器，而只有一组通用寄存器，所以通用寄存器的影子即为该寄存器本身。表 3.16 给出了 SRSCtl 寄存器的格式，及各域的含义。

31–16	15–12	11–10	9–6	5–0
0	ESS	0	PSS	0
16	4	2	4	6

位域	描述
ESS	用于例外的影子寄存器组，在龙芯 GS464 上只能为 0。
PSS	前一个影子寄存器组，在龙芯 GS464 上只能为 0。
0	保留：必须按 0 写入，读时返回 0。

表 3.16: CP0: SRSCtl 寄存器

3.1.15 Cause（13）寄存器

Cause 寄存器是一个 32 位的可读写寄存器，所有关于最近发生的例外的相关信息都存储在这个寄存器中。表 3.17 给出了 Cause 寄存器的格式，及各域的含义。

表 3.17: CP0: Cause 寄存器

31	30	29 28	27	26	25 24	23	22 16	15 8	7	6 2	1 0
BP	TI	CE	DC	PCI	0	IV	0	IP7-IP0	0	ExcCode	0
1	1	2	1	1	2	1	7	8	1	5	2

位域	描述
BD	例外分支延时槽指示位 — 1：最近的例外发生在延时槽中。
TI	时钟中断指示位 — 1：时间中断等待处理。
CE	引发协处理器不可用例外的协处理器单元编号。
DC	计数寄存器禁用位 — 1：计数寄存器禁用。
PCI	性能计数中断指示位 — 1：性能计数器中断等待处理。
IV	中断例外入口位 — 0：通用例外向量（0x180）；1：特殊中断向量（0x200）。
IP	中断指示位：某位为 1 表明对应中断等待。其值将保持不变直到中断撤除。其中 IP0, IP1 是软中断位，可由软件设置与清除。
ExcCode	例外码域（编码及含义见表 3.18）。
0	保留：必须按 0 写入，读时返回 0。

例外码（ExcCode）是一个 5 位的位域；表 3.18 列出了所有例外码的编码及其含义。

表 3.18: Cause 寄存器 ExcCode 域

例外代码	助记符	描述	例外代码	助记符	描述
0	Int	中断	16	IS	栈例外
1	Mod	TLB 修改例外	17	—	保留
2	TLBL	TLB 例外（读取）	18	—	保留
3	TLBS	TLB 例外（存储）	19	DIB	Debug 指令例外
4	AdEL	地址错误例外（读取）	20	DDBS	Debug 存数据例外
5	AdES	地址错误例外（存储）	21	DDBL	Debug 取数据例外
6	IBE	指令总线错误例外	22	—	保留
7	DBE	数据总线错误例外	23	WATCH	Watch 例外
8	Sys	系统调用例外	24	—	保留
9	Bp	断点例外	25	—	保留
10	RI	保留指令例外	26	DBP	Debug 断点例外
11	CpU	协处理器不可用例外	27	DINT	Debug 调试例外
12	Ov	算术溢出例外	28	DSS	Debug 单步例外
13	Tr	陷阱例外	29	—	保留
14	—	保留	30	CacheErr	Cache 错例外
15	FPE	浮点例外	31	—	保留

3.1.16 EPC（14）寄存器

EPC（Exception Program Counter，例外程序计数器）寄存器是一个 64 位可读写寄存器，它的内容是例外处理结束后的程序恢复运行的地址。表 3.19 给出了 EPC 寄存器的格式。

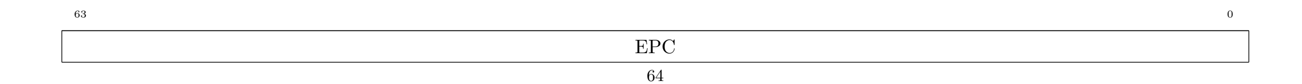


表 3.19: CP0: EPC 寄存器

当同步例外发生时，有两种可能：

- 一般而言，EPC 寄存器的内容为例外发生时的指令虚地址，该指令为导致例外的直接原因；
- 如果例外发生时，指令在分支延时槽中，则 EPC 内容为之前的分支或者跳转指令的虚地址，同时 Cause 寄存器的指令延时位（BD）将被置位。

当 Status 寄存器中的 EXL 位被置 1，即例外在另外一个例外中被触发时，处理器不会重写 EPC 寄存器。

3.1.17 PRid（15）寄存器

PRid (Processor Revision Identifier，处理器修改版本标识) 寄存器是一个 32 的只读寄存器。表 3.20 给出了 PRid 寄存器的格式，及各域的含义。

31–16	15–8	7–0
0	Imp	Rev
16	8	8

位域	描述
Imp	实现版本号。
Rev	修订版本号。
0	保留：必须按 0 写入，读时返回 0。

表 3.20: CP0: PRid 寄存器

版本号由 8 位信息表达，一般写作 `Y.X`，其中 `Y`（7:4 位）为主版本号，而 `X`（3:0 位）为次版本号。龙芯 GS464 核的实现版本号为 `6.3`，修订版本号为 `0.3`。

注意，版本号码可以区分一些处理器的版本，但处理器的任何改动不能保证都体现在 PRid 寄存器中；另一方面，版本号的改动也不能保证必须体现处理器的修改。因此，软件不能过分依赖 PRid 寄存器中的版本号信息来标识处理器。

3.1.18 EBase（15）寄存器

EBase 寄存器是一个 32[2] 位可读写寄存器：它包含有确定例外向量基地址的信息，以及附加的只读处理器号信息。当状态 (Status) 寄存器的 BEV=0，即程序在正常运行状态（而非启动运行时），系统将使用 EBase 寄存器中的例外向量基址域来决定向量入口。具体细节见 6.1.3 节。表 3.21 给出了 EBase 寄存器的格式，及各域的含义。

[2] 32 位的 EBase 寄存器就足够决定例外向量基地址，这是因为出于兼容 32 位系统的需要，例外向量基地址被放在 CKSEG0/CKSEG1 兼容段。这是一种约定，对实际操作不构成任何实质的限制。

31	30	29 … 12	11 … 10	9 … 0
1	0	EBase	0	CPUNum
1	1	18	2	10

位域	描述
[31:30]	MIPS64 架构定义的只读域，用于确认例外向量的地址段。
EBase	例外向量基地址域：与 31:30 位联合指明例外向量基址。
CPUNum	在多核系统中，用于指明处理器号。
0	保留：必须按 0 写入，读时返回 0。

表 3.21: CP0: EBase 寄存器

3.1.19 Config 寄存器

Config 寄存器是一个 64 位寄存器，它存有龙芯 GS464 处理器核中各种配置选择项信息。

- 寄存器位 31:3 所定义的配置选项，由硬件复位时设置，作为只读状态位供软件访问；
- 寄存器位 2:0 与 Cache 配置有关，为可读写位，由软件所控制。复位时这些位是没有定义的，在 Cache 被使用之前由软件来初始化，并在任何改变后重新初始化。

Config 寄存器的初值为 0x0003_0932(Remark: 有问题！)。表 3.22 给出了 Config 寄存器的格式，及各域的含义。

31	30 … 16	15 14 13 12	10 9	7 6		4	3	2 … 0
M	0	BE	AT	AR	MT	0	VI	K0
1	15	4	3	3	1	1	1	3

位域	描述
M	Config1 寄存器存在位 — 置 1 表示存在。
BE	尾端类型位 — 1：大尾端；0：小尾端。
AT	架构类型域 — 0：MIPS32；1：MIPS64(32 位地址段)；2：MIPS64；3：保留。
AR	架构发布版本域 — 0：Release 1；1：Release 2；其他：保留。
MT	内存管理类型域 — 0：无映射；1：标准 TLB；其他：保留。
VI	虚拟指令 Cache 位 — 0：无虚拟指令 Cache；1：有虚拟指令 Cache。
K0	CKSEG0/KSEG0 段一致性算法属性位。
0	保留：必须按 0 写入，读时返回 0。

表 3.22: CP0: Config 寄存器

3.1.20 Config1 寄存器

Config1 寄存器是一个 32 位的只读寄存器。作为 Config 寄存器的附加内容，它含有龙芯 GS464 处理器核的 Cache 配置，和各种其他附加信息。该寄存器在复位时被自动设置；在 GS464 核上，该寄存器的值为 0xFEE3_7193。表 3.23 给出了 Config1 寄存器的格式，及各域的含义。

表 3.23: CP0: Config1 寄存器

位	31	30–25	24–22	21–19	18–16	15–13	12–10	9–7	6	5	4	3	2	1	0
域	M	MMU Size	IS	IL	IA	DS	DL	DA	C2	MD	PC	WR	CA	EP	FP
值	1	1 1 1 1 1 1	0 1 1	1 0 0	0 1 1	0 1 1	1 0 0	0 1 1	0	0	1	0	0	1	1
宽度	1	6	3	3	3	3	3	3	1	1	1	1	1	1	1

<table>
<tr><th>位域</th><th>描述</th></tr>
<tr><td>M</td><td>Config2 寄存器存在位：置 1 表示存在。</td></tr>
<tr><td>MMU Size</td><td>TLB 表项最大索引数（表项总数减 1）。</td></tr>
<tr><td>IS</td><td>Icache 每路组数。<table><tr><td>编码</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>含义</td><td>64</td><td>128</td><td>256</td><td>512</td><td>1024</td><td>2048</td><td>4096</td><td>保留</td></tr></table></td></tr>
<tr><td>IL</td><td>Icache 每组大小（字节）。<table><tr><td>编码</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>含义</td><td>无</td><td>4</td><td>8</td><td>16</td><td>32</td><td>64</td><td>128</td><td>保留</td></tr></table></td></tr>
<tr><td>IA</td><td>Icache 相联方式。<table><tr><td>编码</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>含义</td><td>直接</td><td>2 路</td><td>3 路</td><td>4 路</td><td>5 路</td><td>6 路</td><td>7 路</td><td>8 路</td></tr></table></td></tr>
<tr><td>DS</td><td>Dcache 每路组数。<table><tr><td>编码</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>含义</td><td>64</td><td>128</td><td>256</td><td>512</td><td>1024</td><td>2048</td><td>4096</td><td>保留</td></tr></table></td></tr>
<tr><td>DL</td><td>Dcache 每组大小（字节）。<table><tr><td>编码</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>含义</td><td>无</td><td>4</td><td>8</td><td>16</td><td>32</td><td>64</td><td>128</td><td>保留</td></tr></table></td></tr>
<tr><td>DA</td><td>Dcache 相联方式。<table><tr><td>编码</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>含义</td><td>直接</td><td>2 路</td><td>3 路</td><td>4 路</td><td>5 路</td><td>6 路</td><td>7 路</td><td>8 路</td></tr></table></td></tr>
<tr><td>C2</td><td>2 号协处理器实现位 — 0: 未实现; 1: 实现。</td></tr>
<tr><td>MD</td><td>MDMX ASE 实现位 — 0: 未实现; 1: 实现。</td></tr>
<tr><td>PC</td><td>性能计数器实现位 — 0: 未实现 1: 实现。</td></tr>
<tr><td>WR</td><td>Watch 寄存器实现位 — 0: 未实现 1: 实现。</td></tr>
<tr><td>CA</td><td>MIPS16e 实现位 — 0: 未实现 1: 实现。</td></tr>
<tr><td>EP</td><td>EJTAG 实现位 — 0: 未实现 1: 实现。</td></tr>
<tr><td>FP</td><td>FPU 实现位 — 0: 未实现; 1: 实现。</td></tr>
</table>

3.1.21 Config2 寄存器

Config2 寄存器是一个 32 位的只读寄存器。同样作为 Config 寄存器的附加内容，它含有龙芯 GS464 处理器二、三级 Cache 的配置信息。该寄存器在复位时被自动设置；在 GS464 核上，该寄存器的值为 0x8000_1643。表 3.24 给出了 Config2 寄存器的格式，及各域的含义。

表 3.24: CP0: Config2 寄存器

31	30-28	27-24	23-20	19-16	15-12	11-8	7-4	3-0
M	TU	TS	TL	TA	SU	SS	SL	SA
1	3	4	4	4	4	4	4	4

位域	描述
M	Config3 寄存器存在位：置 1 表示存在。
TU	二级 cache 控制状态位
TS	三级 cache 每路组数。

编码	0	1	2	3	4	5	6	7	8-15
含义	直接	2 路	3 路	4 路	5 路	6 路	7 路	8 路	保留

TL 三级 cache 每组大小（字节）。

编码	0	1	2	3	4	5	6	7	8-15
含义	无	4	8	16	32	64	128	256	保留

TA 三级 cache 相联方式。

编码	0	1	2	3	4	5	6	7	8-15
含义	直接	2 路	3 路	4 路	5 路	6 路	7 路	8 路	保留

位域	描述
SU	二级 cache 控制状态位。
SS	二级 cache 每路组数。

编码	0	1	2	3	4	5	6	7	8-15
含义	64	128	256	512	1024	2048	4096	8192	保留

SL 二级 cache 每组大小（字节）。

编码	0	1	2	3	4	5	6	7	8-15
含义	无	4	8	16	32	64	128	256	保留

SA 二级 cache 相联方式。

编码	0	1	2	3	4	5	6	7	8-15
含义	直接	2 路	3 路	4 路	5 路	6 路	7 路	8 路	保留

3.1.22 Config3 寄存器

Config3 寄存器是一个 32 位的只读寄存器。作为 Config 寄存器的附加内容，该寄存器标记了龙芯 GS464 处理器核一些功能是否实现。在 GS464 核上，该寄存器的值为 0x0000_00A0，在硬件复位时被自动设置。表 3.25 给出了 Config3 寄存器的格式，及各域的含义。

表 3.25: CP0: Config3 寄存器

31	30-11	10	9-8	7	6	5	4	3	2	1	0
M	0	DSPP	0	LPA	VEIC	Vint	SP	0	MT	SM	TL
1	20	1	2	1	1	1	1	1	1	1	1

位域	描述
M	保留：必须按 0 写入，读时返回 0。
0	保留：必须按 0 写入，读时返回 0。

未完待续

表 3.25: CP0: Config3 寄存器（续）

位域	描述
DSPP	DSP ASE 实现指示位。
LPA	大物理地址实现指示位。
VEIC	外部中断控制器实现指示位。
Vint	向量中断实现指示位。
SP	小页面支持实现指示位。
MT	MIPS MTASE 实现指示位。
SM	SmartMIPS ASE 实现指示位。
TL	Trace Logic 实现指示位。

3.1.23 LLAddr 寄存器

LLAddr (Load Link Address) 寄存器是一个 64 位只读寄存器, 用于存放最近发生的 load-linked 指令地址的页帧号 (PFN)。当例外从 ERET 指令返回时，LLAddr 寄存器被清零。表 3.26 给出了 LLAddr 寄存器的格式。

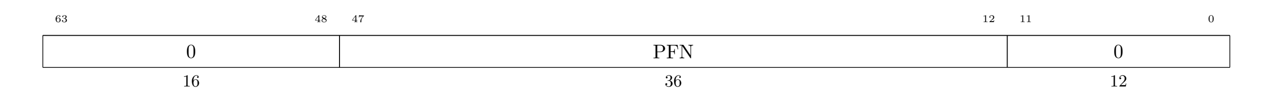


表 3.26: CP0: LLAddr 寄存器

3.1.24 XContext 寄存器

XContext 寄存器是一个 64 位的可读写寄存器。它包含有一个指向页表入口的指针，用于 XTLB 重填处理，处理 64 位地址空间的 TLB 表项加载操作。表 3.27 给出了 XContext 寄存器的格式，及各域的含义。

63–41	40–39	38–4	3–0
PTEBase	R	BadVPN2	0
23	2	35	4

位域	描述
PTEBase	页表入口基地址域：其值为指向内存中当前页表的指针。
R	区域位 — 00: 用户; 01: 管理; 11: 内核 — 虚地址 63:62 位。
BadVPN2	未能有效转换的虚地址虚页号（VPN2）：由硬件在 TLB 例外时写入。
0	保留：必须按 0 写入，读时返回 0。

表 3.27: CP0: XContext 寄存器

操作系统会设置寄存器中的 PTEBase 域。当 TLB 重填例外发生时，例外处理程序该寄存器的内容从页表中加载缺失页信息到 TLB 表项中。35 位的 BadVPN2/VPN2 域对应着引发 TLB 例外的虚拟地址的 47:13 位。位 12 被排除在外，是因为一个 TLB 表项映射到两个页面。当页面大小为 4K 字节时，VPN2 可以直接寻址 8 字节长，成对组织的页表入口数列（PTE array）。对于其他大小的页面，则需要掩去 VPN2 中多余的位以得到正确的页表入口地址。

3.1.25 Diagnostic 寄存器

Diagnostic 寄存器是一个龙芯处理器特有的 64 位可读写寄存器，主要用于控制处理器的一些内部队列和特殊操作。表 3.28 给出了 Diagnostic 寄存器的格式，及各域的含义。

表 3.28: CP0: Diagnostic 寄存器

63–8	7	6	5	4	3	2	1	0
0	W-CAC	W-ISS	S-ISS	S-FET	0	ITLB	BTB	RAS
56	1	1	1	1	1	1	1	1

位域	描述
W-CAC	取消 Wait-cache 操作的限制。
W-ISS	取消 Wait-issue 操作的限制。
S-ISS	取消 Store-issue 操作的限制。
S-FET	取消 Store-fetch 操作的限制。
ITLB	写入 1 时清空 ITLB。
BTB	写入 1 时清空 BTB。
RAS	写入 1 时禁止使用 RAS。
0	保留：必须按 0 写入，读时返回 0。

3.1.26 Debug 寄存器

Debug 寄存器是一个 32 位的可读写寄存器。它包含有最近发生的调试例外或者在调试模式下发生的例外的信息，它同时还控制单步中断，并指明了调试模式的可用资源和其他调试相关的内部状态。

表 3.29 给出了 Debug 寄存器的格式，及各域的含义。注意，该寄存器只有 LSNM 域和 SSt 域可写；而在非调试模式下，则只有 DM 位和 EJTAGver 域可读。系统复位时，Debug 寄存器的初始值为：0x0201_8000。Debug 寄存器中的各位域，只有在调试例外或调试模式下例外发生时才会被更新。

龙芯 GS464 处理器核未实现调试例外时的省电模式。

表 3.29: CP0: Debug 寄存器

31	30	29	28	27–26	25	24–18	17–15	14–10	9	8	7–6	5	4	3	2	1	0
DBD	DM	NoDCR	LSNM	0	CountDM	0	EJTAGver	DExcCode	NoSSt	SSt	0	DINT	DIB	DDBS	DDBL	DBp	DSS
1	1	1	1	2	1	7	3	5	1	1	2	1	1	1	1	1	1

位域	描述
DBD	调试例外指令延迟槽指示位。
DM	调试模式位 — 0: 非调试模式; 1: 调试模式。
NoDCR	DSEG 段存在位 — 1: 不存在; 0: 存在。
LSNM	DSEG 段存在时，存取可用地址位 — 0: DSEG 段; 1: 系统内存。

未完待续

表 3.29: CP0: Debug 寄存器（续）

位域	描述
CountDM	调试模式下 Count 寄存器工作状态位 — 1: 继续计数。
EJTAGver	EJTAG 版本域 — 0: 版本 1/2.0; 1: 2.5; 2: 2.6; 3: 3.1; 4: 保留。
DexcCode	调试模式下的例外原因域。
NoSSt	单步中断指示位 — 0: 支持单步; 1: 不支持。
SSt	单步中断使能位 — 1: 使能单步中断。
DINT	调试中断例外发生位：进入调试模式后自动清零。
DIB	调试指令中断例外发生位：进入调试模式后自动清零。
DDBS	调试数据中断例外发生位：进入调试模式后自动清零。
DBp	调试断点例外发生位：进入调试模式后自动清零。
DSS	调试单步中断例外发生位：进入调试模式后自动清零。
0	保留：必须按 0 写入，读时返回 0。

3.1.27 DEPC 寄存器

DEPC（Debug Exception Program Counter, 调试例外程序计数）是一个 64 位的可读写寄存器，它包含有调试例外处理结束后的继续处理地址。此寄存器由硬件在调试例外或调试模式下的例外时更新。表 3.30 给出了 DEPC 寄存器的格式。

对于精确的调试例外和调试模式下的精确例外，DEPC 寄存器的内容是下面之一:

- 指令虚地址，这是导致例外的直接原因，或者
- 当指令在分支延时槽中，则为之前的分支或者跳转指令的虚地址。同时，Debug 寄存器的指令延时槽指示位 DBD 被置 1。

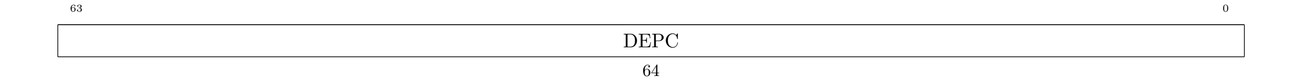


表 3.30: CP0: DEPC 寄存器

3.1.28 PerfCnt 寄存器 (25，0/1/2/3)

龙芯 GS464 处理器定义了四个（两组）性能计数器 (Performance Counter)，他们分别映射到 CP0 寄存器的 25 号的选择子号 0，1，2，3。表 3.1.28 列出了这四个选择子号对应的性能计数器含义。

选择号	0	1	2	3
寄存器	控制寄存器 0	计数寄存器 0	控制寄存器 1	计数寄存器 1

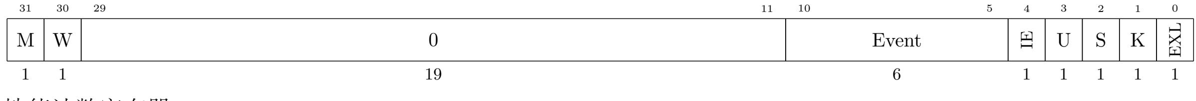
表 3.31: CP0: PerfCnt 控制、计数寄存器选择号列表

这两组计数器格式相同：其中性能控制是 32 位的寄存器，而性能计数器则为 64 位。表 3.32 给出了 PerfCnt 寄存器的格式，及各域的含义。龙芯 GS464 在复位时，PerfCnt 寄存器的两个控制寄存器赋的初始值分别为：

- PerfCnt0 (select 0) = 0xC000_0000
- PerfCnt1 (select 2) = 0x4000_0000

每组计数器都可以独立对一种事件计数，并且在相关的事件域（Event）中对应的可数事件发生时自增。当性能计数器溢出，即首位（63 位）变成 1，则将触发一个中断：Cause 寄存器中的 PCI 位被置 1 （如果有多组计数器，则 PCI 位的值为多组计数器的溢出位取或）。计数器溢出后，无论中断是否被处理，计数都将继续。表 3-26 描述计数使能位的定义。表计数器 0 和计数器 1 各自的事件。

性能控制寄存器:



性能计数寄存器:

64 0
Event Count
64

位域	描述
M	扩展计数器存在位：为 1 则还有另一组计数器。
W	计数寄存器宽度位 — 0: 32 位; 1: 64 位。
K	内核模式位。
S	管理模式位。
U	用户模式位。
EXL	例外级指示位。

表 3.32: CP0: PerfCnt 寄存器

表 3.33: PerfCnt: 计数器 0 事件

事件	信号	描述
0000	Cycles	周期
0001	Brbus.valid	分支指令
0010	Jrcount	JR 指令
0011	Jr31count	JR 指令并且域 rs=31
0100	Imemread.valid & Imemread_allow	一级 I-cache 缺失
0101	Rissuebus0.valid	Alu1 操作已发射
0110	Rissuebus2.valid	Mem 操作已发射
0111	Rissuebus3.valid	Falu1 操作已发射
1000	Brbus_bht	BHT 猜测指令
1001	Mreadreq.valid & Mreadreq_allow	从主存中读
1010	Fxqfull	固定发射队列满的次数

未完待续

表 3.33: PerfCnt: 计数器 0 事件（续）

事件	信号	描述
1011	Roqfull	重排队列满的次数
1100	Cp0qfull	CP0 队列满的次数
1101	Exbus.ex & ExcCode=34,35	Tlb 重填例外
1110	Exbus.ex & ExcCode=0	例外 (??)
1111	Exbus.ex & ExcCode=63 (how come?)	内部例外

表 3.34: PerfCnt: 计数器 1 事件

事件	信号	描述
0000	Cmtbus.valid	提交操作
0001	Brbus.brerr	分支预测失败
0010	Jrmiss	Jr 预测失败
0011	Jr31miss	Jr 且 rs=31 预测失败
0100	Dmemread.valid & Dmemread_allow	一级 DCache 缺失
0101	Rissuebus1.valid ALU2	操作已发射
0110	Rissuebus4.valid FALU2	操作已发射
0111	Duncache_valid & Duncache_allow	访问未缓存
1000	Brbus_bhtmiss	BHT 猜测错误
1001	Mwritereq.valid & Mwritereq_allow	写到主存
1010	Ftqfull	浮点指针队列满的次数
1011	Brqfull	分支队列满的次数
1100	Exbus.ex & Op==OP_TLBPI	ITLB 缺失
1101	Exbus.ex	例外总数
1110	Mispec	载入投机缺失
1111	CP0fwd_valid	CP0 队列向前加载

3.1.29 ErrCtl 寄存器

ErrCtl 是一个 64 位可读写寄存器。龙芯 GS464 将这个 MIPS64 标准里可选的寄存器用于 ECC 校验。表 3.35 给出了 ErrCtl 寄存器的格式。

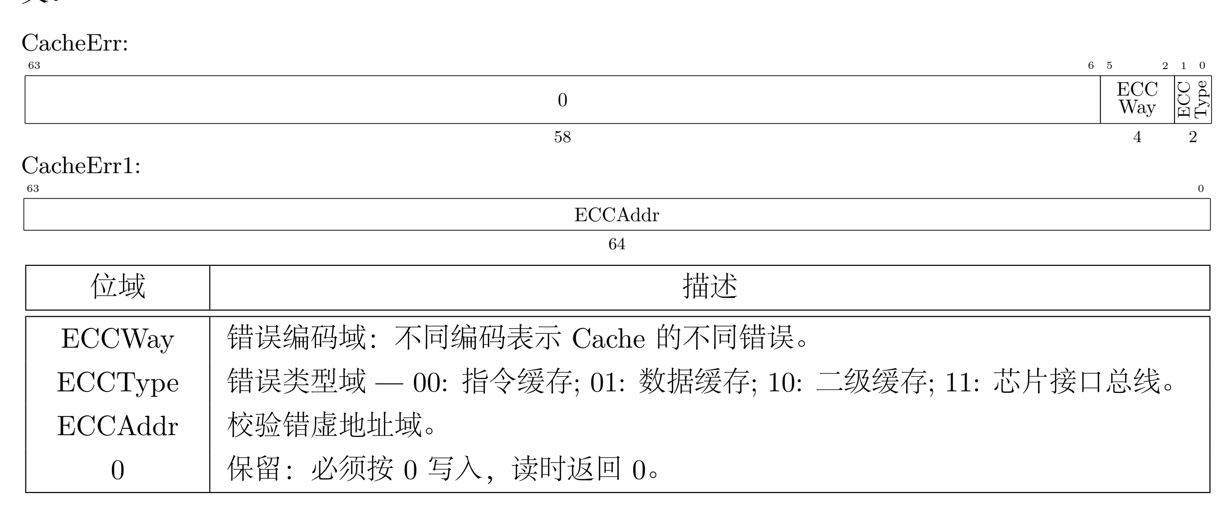
63 ... 8	7 ... 0
0	ECC
56	8

位域	描述
ECC	相关 Cache 的一个双字校验码
0	保留：必须按 0 写入，读时返回 0。

表 3.35: CP0: ErrCtl 寄存器

3.1.30 CacheErr、CacheErr1 寄存器

CacheErr 和 CacheErr1 是一对 64 位可读写寄存器。MIPS64 标准里定义的 ECC 校验是由软硬件共同完成：硬件负责检查错误，并将错误相关的内容保存在 CacheErr 和 CacheErr1 寄存器中，然后触发 Cache 错例外由软件来纠正错误。表 3.36 给出了这两个寄存器的格式，及各域的含义。

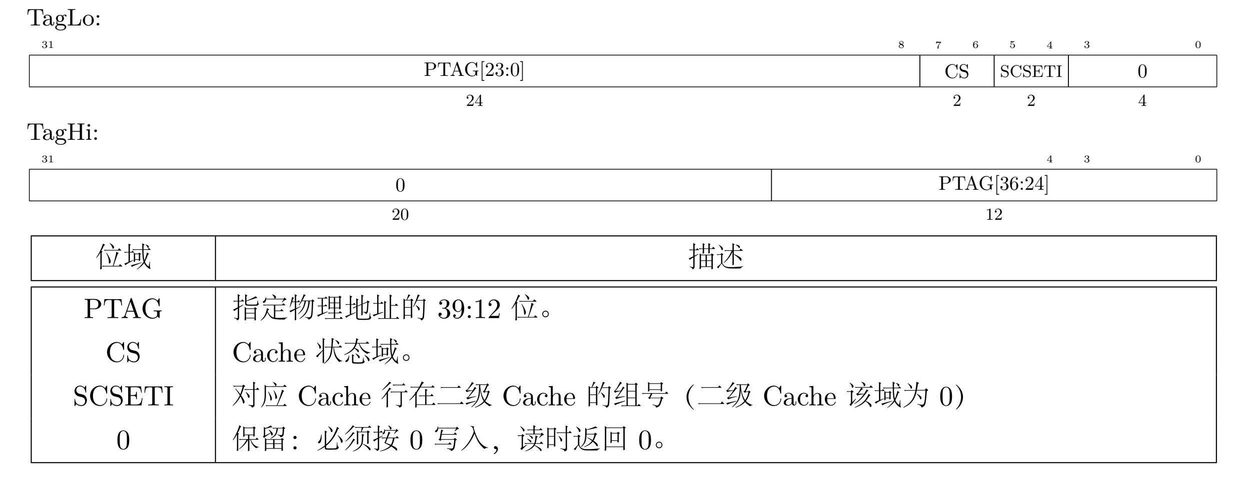


位域	描述
ECCWay	错误编码域：不同编码表示 Cache 的不同错误。
ECCType	错误类型域 — 00: 指令缓存; 01: 数据缓存; 10: 二级缓存; 11: 芯片接口总线。
ECCAddr	校验错虚地址域。
0	保留：必须按 0 写入，读时返回 0。

表 3.36: CP0: CacheErr、CacheErr1 寄存器

3.1.31 TagLo、TagHi 寄存器

TagLo 和 TagHi 寄存器是一对 32 位可读写寄存器，用于保存一、二级 Cache 的标签和状态，并将在使用 CACHE 和 MTC0 指令时将其内容写入 Tag 寄存器。 3.37 显示了这两个寄存器用于一级 Cache（P-Cache）操作时的寄存器格式, 及相应 TagLo 和 TagHi 寄存器各域的定义。



位域	描述
PTAG	指定物理地址的 39:12 位。
CS	Cache 状态域。
SCSETI	对应 Cache 行在二级 Cache 的组号（二级 Cache 该域为 0）
0	保留：必须按 0 写入，读时返回 0。

表 3.37: CP0: TagLo、TagHi 寄存器

3.1.32 DataLo、DataHi 寄存器

DataLo 和 DataHi 是一对 64 位只读寄存器，用于 Cache 数据队列交互和诊断。CACHE 指令的 IndexLoadTag 操作将读取相应数据到 DataLo 或 DataHi 寄存器。表 3.38 给出了 DataLo 和 DataHi 寄存器的格式。

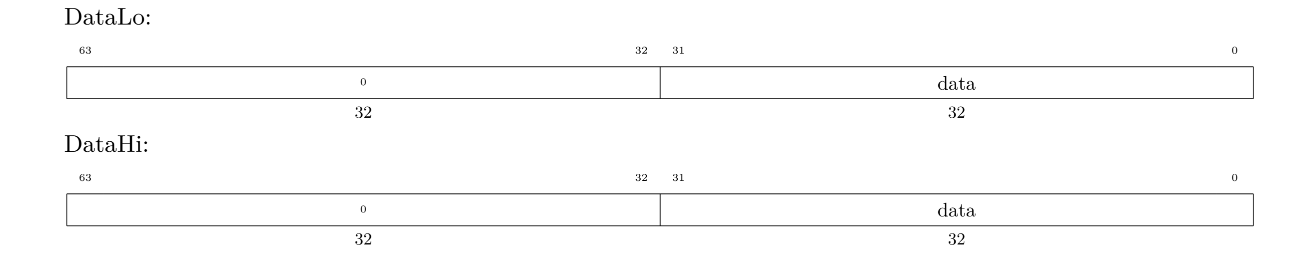


表 3.38: CP0: DataLo、DataHi 寄存器

3.1.33 ErrorEPC 寄存器

ErrorEPC 是一个 64 位可读写寄存器。与 EPC 寄存器类似，ErrorEPC 寄存器含有错误例外后，指令重新开始执行的虚拟地址。除了 ECC 和奇偶错误例外之外，它还用于，在复位、软复位、和不可屏蔽中断（NMI）例外发生时，存储程序返回的地址。表 3.39 显示了 ErrorEPC 寄存器的格式。

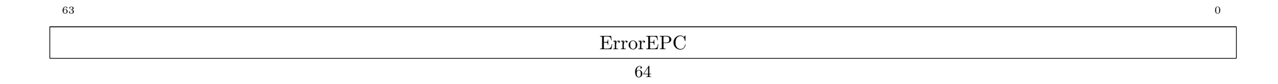


表 3.39: CP0: ErrorEPC 寄存器

3.1.34 DESAVE 寄存器

DESAVE 寄存器是一个 64 位可读写寄存器。它是一个功能简单的暂存器，用于在处理 Debug 异常处理时保存一个通用寄存器的值。DESAVE 寄存器的格式显示在表 3.1.34 中。

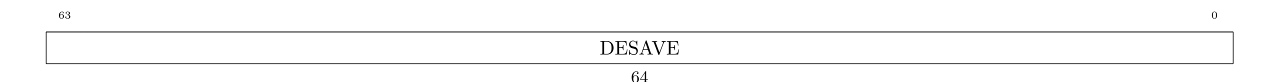


表 3.40: CP0: DESAVE 寄存器

3.2 40 vs 48 位物理地址

第四章　内存管理

龙芯 GS464 处理器核内置了一个功能完备的内存管理单元（MMU）来实现从虚拟地址到物理地址的转换。本章主要讲述处理器核在不同模式下的虚拟地址空间，以及地址转换实现的具体细节。

4.1　快速查找表 TLB

虚拟地址到物理地址的转换通常是在 0 号协处理器的帮助下，通过快速查找表 (Translation Lookaside Buffer, 简称 TLB) 来实现的。注意，某些特定的地址段不需要经过 TLB 就可以进行地址转换，比如 CKSEG0 和 CKSEG1 内核地址空间段，简单地屏蔽掉这些段的虚拟地址的某几位就可以得到它们对应的物理地址，所以它们也被称为非映射段。最重要的 TLB 是混合 TLB (Joint TLB，简称 ITLB)，它可以同时为数据和指令的地址转换服务。另外，龙芯 GS464 处理器核包含有独立的指令 TLB (Instruction TLB，简称 ITLB) 以缓解对 JTLB 的竞争。

4.1.1　混合 TLB

为了能够快速地进行虚拟地址到物理地址的映射，龙芯 GS464 处理器核采用了较大的，全相联映射机制的 JTLB 用于指令和数据的地址映射。JTLB 按偶奇表项成对组织，把虚拟地址空间和地址空间标识符映射到物理地址空间。在默认的情况下，JTLB 有 64 个偶奇对表项，可以最大映射 128 个页面。

GS464 核使用了两种机制来协助控制映射空间的大小和实现对内存不同区域的替换策略。第一，内核的页面大小可变：页面大小的可能取值为在 4KB 到 16MB 之间按 4 的倍数递增的各个级数。映射页的大小记录在 CP0 寄存器 PageMask 中，这个值在每写一个新的 TLB 表项时，将同时载入 TLB 中。龙芯 GS464 处理器在同一时刻，只支持一种页面大小。龙芯系列处理器核有计划提供同一运行时刻不同大小页面的支持，这样允许操作系统对特定应用使用不同的映射以提高效率：例如，当选择适当大小，帧缓冲区就可以只用一个 TLB 表项来进行内存映射。

第二，处理器核在 TLB 缺失的时候将采用随机替换的策略来选择要被替换的 TLB 表项。龙芯系列处理器核提供了 TLB 的锁定功能（通过 CPO 的 Wired 寄存器），这样操作系统可以把一定数量的页面驻留在 TLB 中，而不致于被随机替换出去。这种机制有利于提高操作系统性能，避免死锁；也为实时系统的某些关键软件提供了特定入口。

同时，JTLB 还维护每个页面的 Cache 一致性属性，对于映射的每个页面，JTLB 有特定的缓存代码位域来标识其 Cache 一致性属性。关于 Cache 一致性属性的更多具体信息，请参见5.5 节。

4.1.2 指令 TLB

龙芯 GS464 处理器核的指令 TLB（ITLB）有 16 个表项，它最大限度地减少了对 JTLB 的竞争，有效地在映射时间关键路径上避免了使用一个大的相联阵列进行地址映射，降低了处理器功率。最重要的是，ITLB 的存在使得指令地址映射和数据地址映射得以并行执行，从而提高了处理核的性能。ITLB 的内容是 JTLB 的子集，而且每个 ITLB 表项只能映射一页：页面大小由 PageMask 寄存器来指定。当 ITLB 查找脱靶时，将从 JTLB 中查找相应的表项，并随机选择一个 ITLB 表项进行替换。ITLB 的操作对普通用户是完全透明的。

注意，GS464 处理器没有提供任何机制保证 ITLB 与 JTLB 的一致性：如果 JTLB 被修改时要求 ITLB 也要修改，则需要使用核心态指令刷新 ITLB，否则 ITLB 可能保持旧值。MIPS 兼容性一章，9.2.2 节对 ITLB 刷新的具体细节有详细讲述。

4.1.3 命中和脱靶

如果虚拟地址与 TLB 中某个表项的虚拟地址一致（即 TLB 命中），则页帧号（VPN）将从 TLB 中取出，并和地址偏移（offset）连接组成物理地址。如果虚拟地址与 TLB 中任何表项的虚拟地址都不一致（即 TLB 脱靶），则 CPU 将产生一个例外，并由例外处理程序根据内存的页表内容填入缺失的 TLB 表项。软件既可以指定覆盖某个具体的 TLB 表项，也可以使用硬件提供的机制随机写 TLB 表项。

由于龙芯 GS464 只支持一种地址模式 — 64 位虚地址模式，所以 TLB 重填例外由 XTLB 重填向量来处理。同时，GS464 处理器核的 XTLB 重填向量与 32 位模式下 TLB 的重填向量有相同的例外入口地址。

4.1.4 多项命中

与早期的 MIPS 处理器设计不同，龙芯 GS464 处理器核对 TLB 中虚地址与不只一个表项的虚地址一致的情况，没有提供任何的探测和禁用机制。多项命中并不会物理地破坏 TLB，因此任何探测机制都是不必要的。在 GS464 核上，TLB 多项命中的结果为未定义（undefined），所以操作系统有责任不让这种情况发生。

4.2 虚拟地址空间

龙芯 GS464 处理器核有 3 种工作模式：内核、管理和用户模式。这三种模式的处理器优先级依次降低，同时也有各自不同的地址空间映射。

- 内核模式（最高优先级）：处理器可以访问和改变任何寄存器，操作系统的最内层内核运行在该模式下；
- 管理模式：处理器的优先级降低，操作系统的一些不太关键的部分运行在该模式；
- 用户模式（最低优先级）：用户程序运行在该模式下，这个模式使得不同用户不致互相干扰。

这三种模式的切换主要是由 CP0 的状态寄存器的 KSU 位来控制的。此外，当任何例外发生时，这时状态寄存器的 ERL 位（错误级）或 EXL 位（例外级）被置位，处理器也会被强制切换到内核模式。表 3.14 列出了三种模式切换时，各位域的置位情况（空的表项不重要，可以忽略）。

每种工作模式对应不同的虚拟地址空间。以下小节将详细介绍这三种不同的虚拟地址空间：用户地址空间、管理地址空间和内核地址空间。注意，即使所有虚拟地址空间都是 64 位的。在 48 位物理地址模式下（PABITS = 48），龙芯 GS464 的最大支持的虚拟地址位数为 48 （SEGBITS = 48）。所以，GS464 的地址空间的最大段为 256T (2^{48}) 字节。

4.2.1 用户地址空间

图 4.1 给出了用户模式下，用户虚拟地址空间的示意图。在用户模式下，只有一个称为用户段的单独、统一的虚拟地址空间。

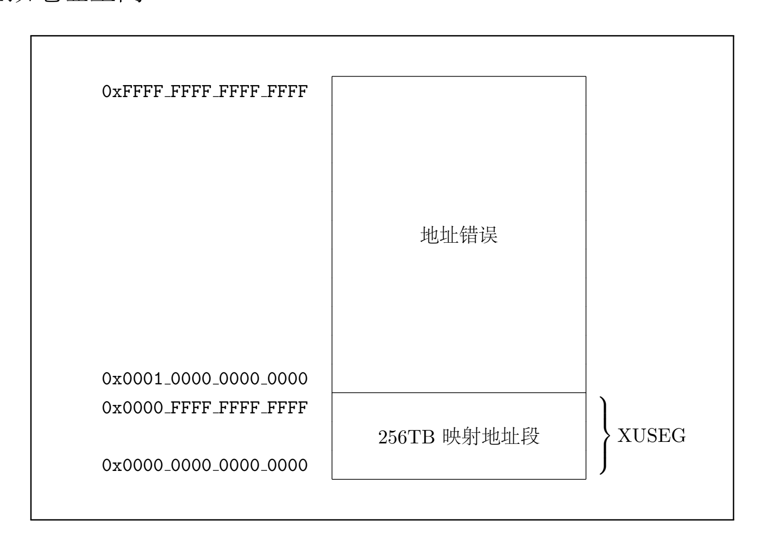


图 4.1: 用户模式下用户虚拟地址空间

- 64 位用户模式用户地址段（XUSEG）。

  用户地址段的大小为 256T 字节，地址从 0 开始，即用户模式下，所有可用的虚拟地址的 63 位到 48 位必须都为 0，任何一个超出这个范围的地址访问都将导致地址错误例外。当前活动的用户进程驻留在该段中。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。该段的低端 2G 与 32 位模式的 USEG 兼容。

  该段在用户模式、管理模式、内核模式下都可以访问，而且在不同模式下，有相同的映射方式和 Cache 访问方式。

4.2.2 管理地址空间

管理模式是为分层结构的操作系统设计的：在一个分层结构的操作系统中，只有真正的内核运行在内核模式下，操作系统的其余部分运行在管理模式下；而管理地址空间提供了管理模式下程序访问的代码和数据空间。在管理模式和内核模式下，都可访问管理地址空间。图 4.2 给出了管理模式下，管理地址空间的示意图。

- 64 位管理模式用户地址段（XSUSEG）

  在管理模式下，当访问地址的最高两位（63 和 62 位）为 00_2 时，程序使用一个名字为 XSUSEG 的虚拟地址空间段：该段覆盖了当前用户地址空间 XUSEG 段。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

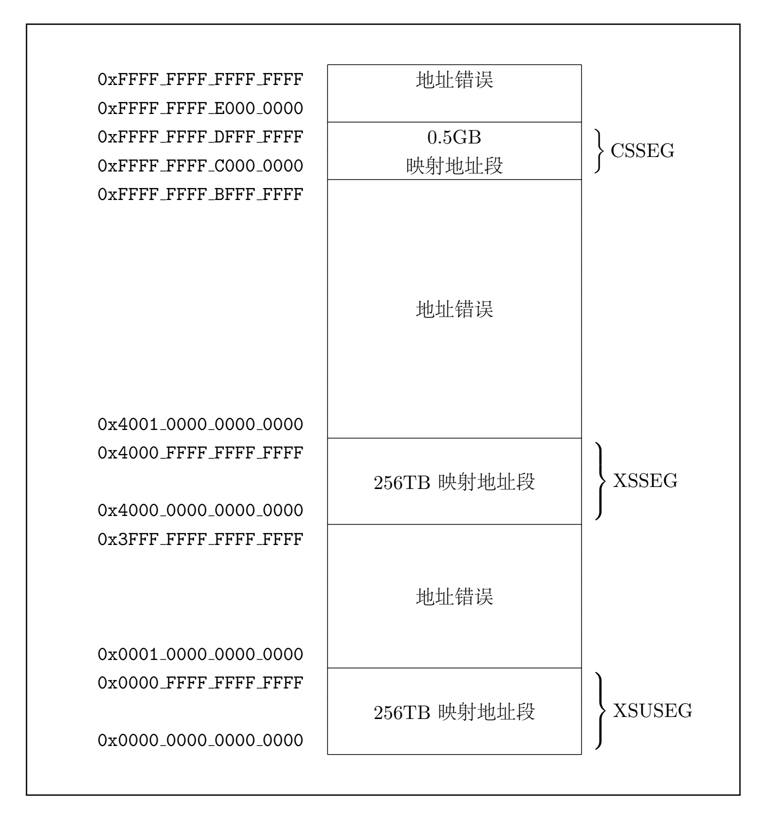


图 4.2: 管理模式下虚拟地址空间分布

- 64 位管理模式当前管理地址段（XSSEG）

  在管理模式下，当 64 位地址的最高两位（第 63 和第 62 位）为 01_2 时，程序使用一个名字为 XSSEG 的管理虚拟地址段。该虚拟地址段被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

- 64 位管理模式，独立管理地址段（CSSEG）

  在管理模式下，当 64 位地址的最高两位（第 63 和第 62 位）为 11_2 时，程序使用一个名字为 CSSEG 的独立管理虚拟地址段。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

  该段地址空间从 0xFFFF_FFFF_C000_0000 开始，到 0xFFFF_FFFF_DFFF_FFFF 结束，大小为 0.5G。该地址段中的寻址与 32 位模式下 SSEG 段是兼容的。

4.2.3 内核地址空间

图 4.3 给出了内核模式下，内核地址空间的示意图。

- 64 位内核模式用户地址段（XKUSEG）

  在内核模式下，当 64 位虚拟地址的最高两位为 00_2 时，程序使用一个名字为 XKUSEG 的虚拟地址空间。该地址段覆盖了用户地址段 XUSEG。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

- 64 位内核模式当前管理地址段（XKSSEG）

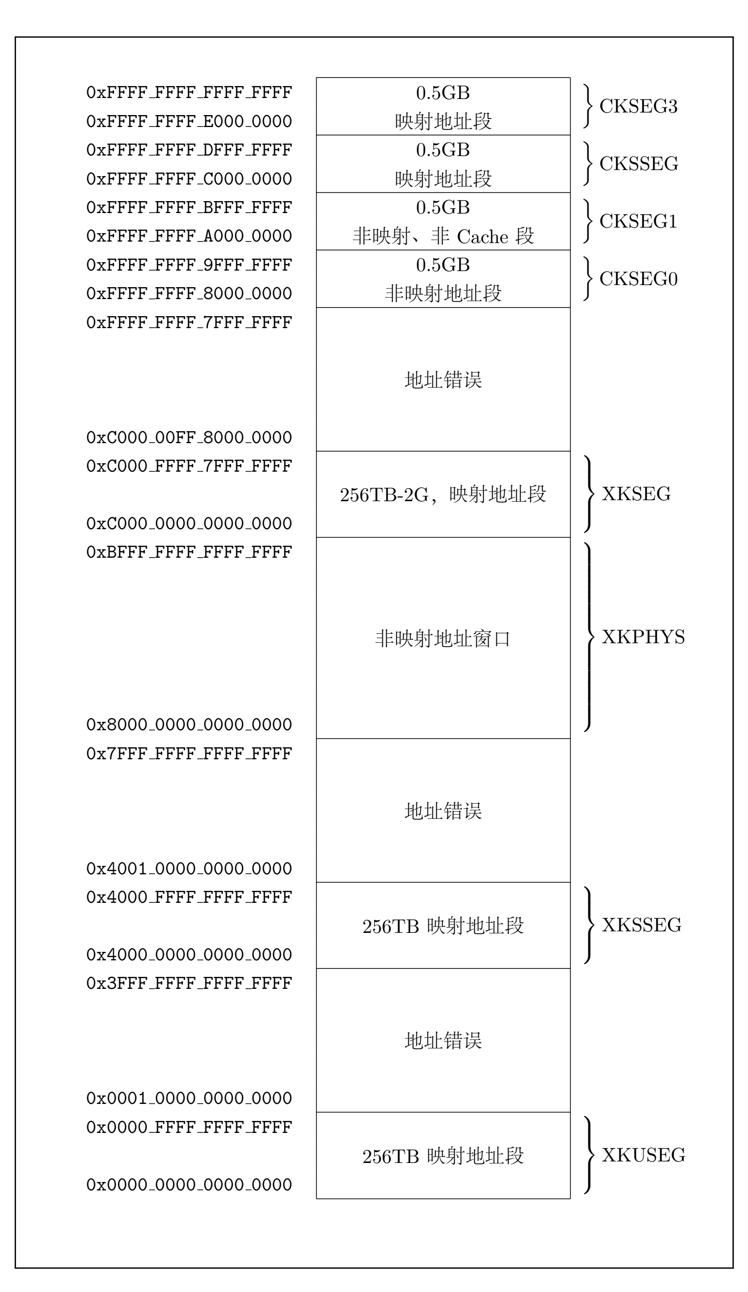


图 4.3: 内核模式虚拟地址空间分布

在内核模式下，当 64 位虚拟地址的最高两位为 01_2 时，程序使用一个名字为 XKSSEG 的虚拟地址空间。该地址段覆盖了当前管理地址空间段 XSSEG。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址。

- 64 位内核模式物理地址窗口（XKPHYS）

  在内核模式下，当 64 位虚拟地址的最高两位为 10_2 时，程序使用一个名字为 XKPHYS 的虚拟地址空间窗口。该地址空间窗口是 8 个 256T 大小的地址段的集合。对这些地址空间的访问不需要 TLB 进行地址变换，而是直接将虚拟地址的 47:0 位作为物理地址。虚地址的第 61 到第 59 位为地址的 Cache 一致性属性。对该地址窗口中任何 58 到 48 位不为 0 的存储访问都将引起地址错例外。从另一个方面说，每一个 Cache 一致性属性都对应着一个 XKPHYS 中的 256T 的地址段。

  该地址窗口是对 32 位模式下，KSEG0 和 KSEG1 概念的扩展。这 8 个地址段都映射到相同的物理地址，但是各自有不同的 Cache 策略。

- 64 位内核模式内核地址空间（XKSEG）

  这是 64 位地址的最高两位为 11_2 的地址空间段之一。该段从开始，大小为。此时虚拟地址被扩展，加上 8 位的 ASID 域，形成一个系统中唯一的虚拟地址；

- 64 位内核模式兼容地址段（CKSEG0，CKSSEG1，CKSSEG，CKSEG3）

  这是四个 64 位地址的最高两位为 11_2 的地址段，并且所有虚拟地址的 61 到 31 位所有位亦为 1。具体使用那个 512M 地址段根据地址的 30、29 位决定：

  – CKSEG0：不经过 TLB 映射，与 32 位模式 KSEG0 兼容。Config 寄存器的 K0 域控制是否通过页面的 Cache 的一致性属性；

  – CKSEG1：不经过 TLB 也不经过 Cache，与 32 位模式 KSEG1 兼容；

  – CKSSEG：当前管理虚拟地址空间，与 32 位模式 KSSEG 兼容；

  – CKSEG3：内核虚拟地址空间，与 32 位模式 KSEG3 兼容。

4.3 虚拟物理地址转换

本节叙述虚拟地址经过 TLB 转换为物理地址空间的细节。这种地址转换是在 0 号协处理器（CP0）的帮助下，通过 TLB 实现的。下面表格列出了所有与地址转换相关的 CP0 寄存器，而所有与 TLB 操作有关的指令见表 2.4。

寄存器号	寄存器名	寄存器号	寄存器名	寄存器号	寄存器名
0	Index	5	PageMask	16	Config
1	Random	6	Wired	17	LLAddr
2	EntryLo0	10	EntryHi	28	TagLo
3	EntryLo1	15	PRID	29	TagHi

表 4.1: 地址转换相关 CP0 寄存器

4.3.1 TLB 表项的格式

图 4.4 给出了 TLB 表项的位域细节。每个 TLB 表项是一个 256 位的数据结构，这些位域分别对应着 CP0 寄存器 PageMask，EntryHi，EntryLo0，和 EntryLo1（从高位到低位的顺序）。关于这些 CP0 寄存器的详细介绍见第三章的具体各节。TLB 表项的格式和这些寄存器的位域基本对应，几个细微的差别有：

- PageMask 是一个 32 位的寄存器，而在 TLB 表项中对应着 64 位：TLB 有更长的保留域；
- TLB 表项中对应 EntryHi 的部分有一个 Global 域（G 位），而 EntryHi 寄存器没有，该位作为保留域出现；
- EntryLo0 和 EntryLo1 各有一个 Global 位，而它们在 TLB 表项的相应位置被保留。这两个 Global 位被合并为一个单独的 G 位，其位置在 EntryHi 的对应部分。

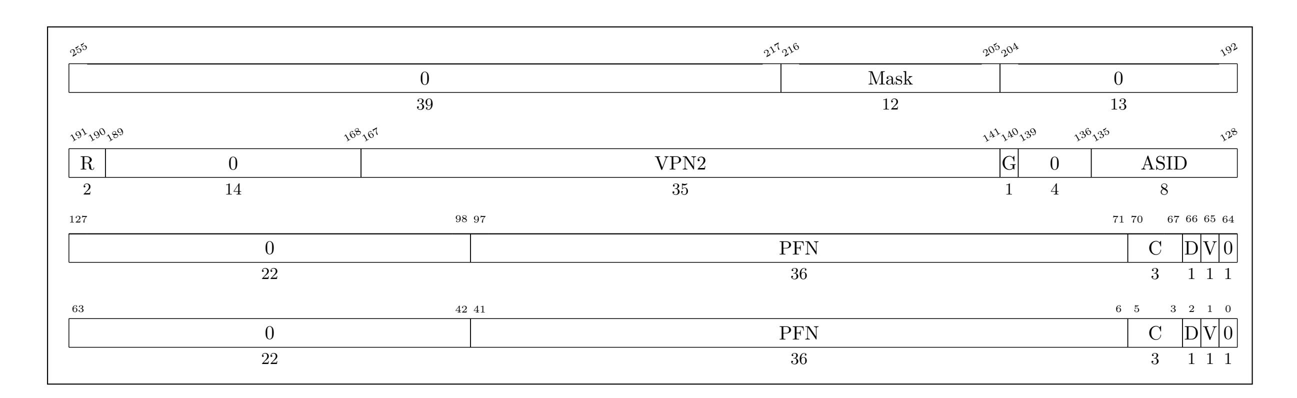


图 4.4: TLB 表项格式

TLB 页项中同时包含有映射页面的 Cache 一致性属性位（C）信息。关于 Cache 一致性属性位的更多细节，参见 5.5 节。

4.3.2 地址转换细节

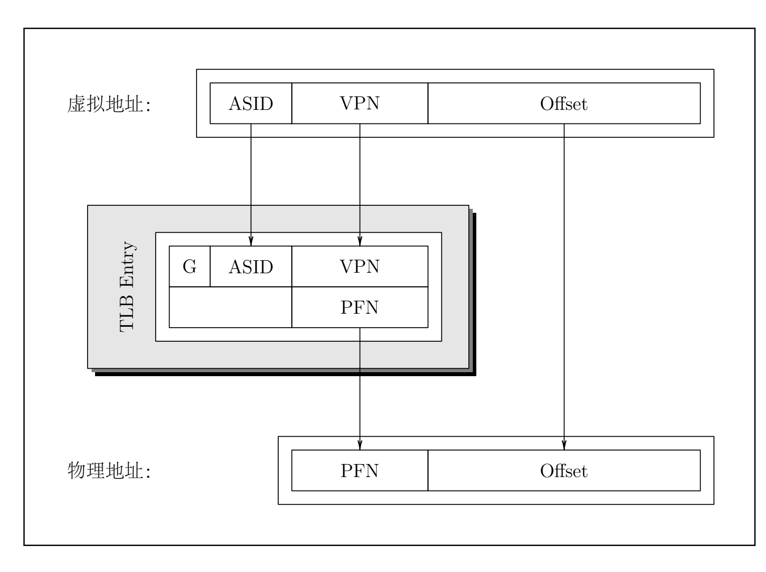


图 4.5: 虚拟、物理地址转换概览

图 4.5 给出了一个简单的虚拟地址到物理址转换的示意图。进行虚实地址转换时，首先将比较处理器给出的虚拟地址和 TLB 中存放的虚拟地址。当虚页号（VPN）等于某个 TLB 表项的 VPN 域，并且如果下面两种情况中的任何一种成立：

- TLB 表项的 Global 位为 1；
- 两个虚拟地址的 ASID 域一样。

那么 TLB 就命中了，页帧号（PFN）将从 TLB 中取出，并与页内偏移量 Offset 合并，形成物理地址。页内偏移量 Offset 在虚实地址转换的过程中不经过 TLB。如果两个条件皆不满足，那么 CPU 会产生 TLB 失效异常，软件将需要页表中存放的信息重新填写 TLB。从地址转换的角度，可以认为 8 位的地址空间标识符（ASID）扩展了虚拟地址，该措施降低了上下文切换时进行 TLB 刷新的频率。

图 4.6 给出了一个更详细的 64 位模式下的地址转换示意图，当页面大小分别为 16K 和 16M 时的情形。

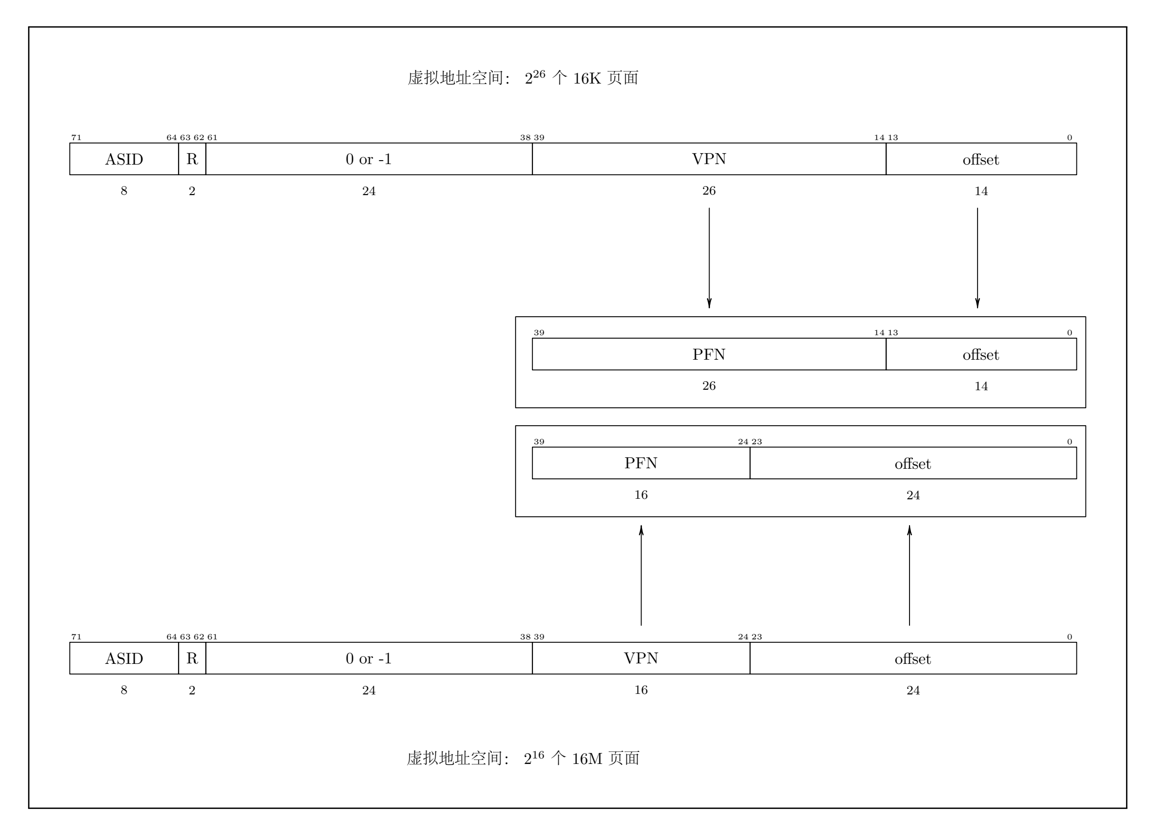


图 4.6: 64 位模式地址转换：16K 和 16M 页面

- 图的上半部分显示了页面大小为 16K 字节的情况，页内偏移量 Offset 占用虚拟地址中的 14 位，虚拟地址中剩下的 26 位为虚页号 VPN，用于索引 64M 个页表表项；
- 图的下半部分显示了页面大小为 16M 字节的情况，页内偏移量 Offset 占用虚拟地址中的 24 位，虚拟地址中剩下的 16 位为虚页号 VPN，用于索引 64K 个页表表项。

图中的 R 域用于选择不同的地址空间：用户、管理或核心地址空间。这些例子示意的是 40 位物理地址模式下的地址转换。当 ELPA 使能时，FPNX 位也将扩展进来和 FNX 一起表示物理高地址；虚地址的 VPN 也会被相应扩展。

在虚地址到物理地址转换时，除了根据页掩码（PageMask）的值将虚地址的高位（即 VPN）和 TLB 项的虚页号进行匹配比较外，还会将虚地址的 8 位 ASID（如果全局位 G 没有设置）和 TLB 项的 ASID 进行比较，看是否匹配。如果有匹配项，还将从匹配的 TLB 表项中取出物理地址和访问控制位（C，D 和 V）。对一个有效的地址转换来说，匹配的 TLB 项的 V 位必须设置，但是在匹配比较时不考虑 V 位的值。如果没有任何一 TLB 项匹配虚地址，则引发一个 TLB 重填例外。如果访问控制位（D 和 V）指示访问不是合法的，引发一个 TLB 修改或者 TLB 无效例外。如果 C 位等于 011_2，被检索到的物理地址通过 Cache 访问内存，否则不通过 Cache。图 4.7 给出了一个完整的 TLB 地址转换的流程。

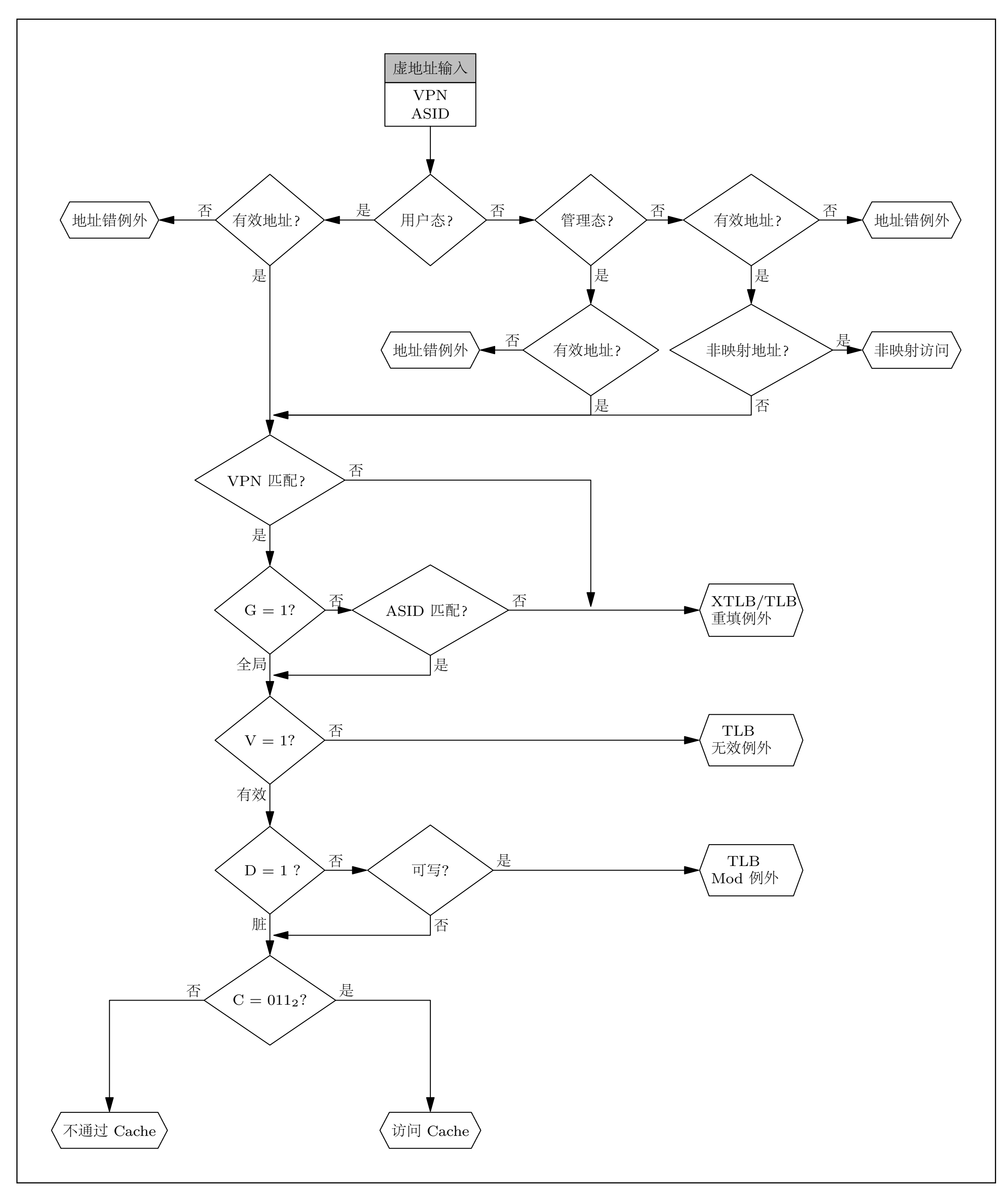


图 4.7: TLB 地址转换流程图

4.3.3 代码例子

第一个例子是如何配置 TLB 表项来映射一对 4KB 的页面。实时系统的内核大多都这么做，这种简单的内核 MMU 只用于进行内存保护，所以静态映射就足够了，在所有静态映射的系统中所有 TLB 例外都被作为是错误条件（不可访问）。

```
mtc0 r0, C0_WIRED                    # 所有表项可以被随机替换
li r2, (VPN2<<13)|(ASID & 0xFF);
mtc0 r2, C0_ENHI                     # 设置虚拟地址
li r2, (EPFN<<6)|(C<<3)|(D<<2)|V<<1|G)
mtc0 r2, C0_ENLO0                    # 设置偶数页的物理地址
li r2, (OPFN<<6)|(C<<3)|(D<<2)|V<<1|G)
mtc0 r2, C0_ENLO1                    # 设置奇数页的物理地址
li r2, 0                             # 设置页面大小为 4KB
mtc0 r2, C0_PAGEMASK
li r2, index_of_some_entry           # needed for tlbwi only
mtc0 r2, C0_INDEX                    # needed for tlbwi only
tlbwr                                # 随机写 TLB 表项
```

一个完备的虚拟存储操作系统（如 UNIX），用 MMU 进行内存保护，并进行主存和大容量存储设备的换页。这个机制使程序可以访问更大的存储设备而不仅仅局限于系统物理分配的空间。这个依赖于请求调页的机制需要动态页面映射。动态映射通过一系列不同类型的 MMU 例外实施，TLB 重填是这种系统中最常见的例外。下面是一个可能的 TLB 重填例外控制。

```
refill_exception:
        mfc0 k0, C0_CONTEXT
        sra  k0, k0, 1          # index into the page table
        lw   k1, 0(k0)          # read page table
        lw   k0, 4(k0)
        sll  k1, k1, 6
        srl  k1, k1, 6
        mtc0 k1, C0_TLBLO0
        sll  k0, k0, 6
        srl  k0, k0, 6
        mtc0 k0, C0_TLBLO1
        tlbwr                   # write a random entry
        eret
```

这个例外控制处理非常简单，因为它的经常执行会影响系统性能，这就是 TLB 重填例外分配独立的例外向量的原因。这段代码假设需要的映射在主存储器的页表中已经建立起来了。如果没有建立起来，那么在 ERET 指令后将发生 TLB 失效例外。TLB 失效例外很少发生，这是有益的，因为它必须计算期望的映射，并可能需要从后援存储器中读取部分页表。TLB 修改例外用于实现只读页面和标记进程清除代码需要修改的页。为了保护不同的进程和用户不受相互的干扰，虚拟存储操作系统通常在用户模式执行用户程序。下面的例子表示如何从内核模式进入用户模式。

```
mtc0 r10, C0_EPC                       # r10: holds desired usermode address
```

```
mfc0 r1, C0_SR                           # get current value of Status register
and  r1, r1, ~(SR_KSU || SR_ERL)         # clear KSU and ERL field
or   r1, r1, (KSU_USERMODE || SR_EXL)    # set usermode and EXL bit
mtc0 r1, C0_SR
eret                                     # jump to user mode
```

4.4 物理地址空间分布

龙芯 2G 的地址空间按照地址的高位均匀分布到各个结点。48 位地址的高 4 位 [47:44] 对应地址空间所在的结点编号，龙芯 2G 结点号为 0，每个结点拥有固定的 44 位地址空间。而在结点内 44 位的地址空间又进一步划分为 8 个 41 位的地址空间，采用 41 位的空间主要是由于一个端口可能会接两个 HT 控制器，而每个 HT 需要 40 位的地址空间。

第五章　CACHE 的组织和操作

5.1　Cache 概述

龙芯 GS464 使用了三种独立的 Cache：

- 一级指令 Cache：64KB 容量，Cache 行大小为 32 字节，四路组相联结构；
- 一级数据 Cache：64KB 容量，Cache 行大小为 32 字节，四路组相联结构，采用写回（Write-back）策略；
- 二级混合 Cache：1M 片上 Cache，行大小为 32 字节，四路组相联的结构，写回策略，通过 128 位 AXI 总线和处理器核通讯。

定点 (ALU) 访问一次一级 Cache 需要 3 个时钟周期，浮点 (FPU) 访问一次一级 Cache 需要 4 个时钟周期。一级 Cache 的读、写和重填通路都是 128 位。每个一级 Cache 都有它们自己的数据通路，所以处理核可以同时访问两个一级 Cache。

二级 Cache 只有在一级 Cache 脱靶时才被访问，这会增加至少 14 个周期的代价。同时由于龙芯 GS464 处理器核需要通过交叉开关和二级 Cache 通讯，还会增加额外的 6 拍。二级 Cache 数据通路为 256 位，在一级 Cache 访问脱靶时，将以每个时钟周期 128 位数据的速度对一级 Cache 进行回填。

一级 Cache 采用虚地址索引和物理地址标志，而二级 Cache 的索引和标志采用的都是物理地址。一级 Cache 的虚地址索引可能会引起 Cache 重影（Cache aliasing）问题。目前龙芯 GS464 需要操作系统通过设置较大的页面（$\geq$ 16K）来避免这个问题，而在将来则通过硬件方式解决。

龙芯 GS464 采用了基于目录的 Cache 一致性协议来保证每个处理器核和 IO 的写操作能被其它处理器核及 IO 设备观察到。这个一致性目录由二级 Cache 维护，对二级 Cache 中的每个 Cache 行用 32 位的向量记录了一级 Cache（包括指令和数据 Cache）的访问信息。从而在硬件上保证了一级指令 Cache、一级数据 Cache、二级 Cache 以及 HT 外部设备之间的 Cache 一致性，而不需要软件的介入。

5.1.1　非阻塞 Cache

龙芯 GS464 实现了非阻塞 Cache 技术，即通过通过允许 Cache 脱靶的访存操作后面的多个访存操作（这些后续访存操作也可能 Cache 脱靶或命中）继续进行，以提高系统的整体性能。在一个阻塞 Cache 的设计中，当发生 Cache 脱靶时，处理器必须暂停后续的访存操作，并开始一个存储周期，取出被请求的数据，将其填入 Cache 中，然后再恢复执行。这个操作过程会占用较多的时钟周期。在非阻塞 Cache 设计中，Cache 并不会因为脱靶而暂停，从而提高了处理器性能。

龙芯 GS464 支持多重脱靶：它的非阻塞队列最大支持连续 24 次 Cache 脱靶的访存操作。当一级 Cache 脱靶时，处理器会检查二级 Cache，若二级 Cache 仍然脱靶，则需要访问主存储器。

龙芯 GS464 中的非阻塞 Cache 结构能有效的使用循环展开和软件流水。从另一个角度看，为了尽大限度地发挥 Cache 的优势，在使用访存数据的指令之前，软件应尽可能早的执行相应的取数操作。针对那些需要顺序存取的 I/O 系统，龙芯 GS464 的默认设置是采用阻塞式的 Uncached 访问方式。

龙芯 GS464 提供了预取指令，可以通过 load 到 0 号定点寄存器的方式来将数据预取到一级数据 Cache。此外龙芯 GS464 中的 DSP 引擎可以将内存或 IO 中的数据预取到二级 Cache 中。（Remark: 什么地方讲了 DSP 引擎？）

5.1.2 替换策略

GS464 核的一级 Cache 和二级 Cache 均采用随机替换算法，但同时对二级 Cache 提供了锁机制。通过配置锁窗口寄存器，可以确保最多 4 个被锁住的区域不被替换出二级 Cache （相关寄存器及具体配置方法参见【龙芯 3A 用户手册】《二级 Cache》一章）。

综上所述，表 5.1 总结了所有关于这三个 Cache 的详细参数列表。

表 5.1: Cache 参数

	指令 Cache	数据 Cache	二级 Cache
Cache 大小	64KB	64KB	1MB
相联度	4 路组相联	4 路组相联	4 路组相联
替换策略	随机法	随机法	随机法（可锁定）
行大小	32 字节	32 字节	32 字节
索引 (Index)	虚地址 13:5 位	虚地址 13:5 位	物理地址 17:5 位[1]
标志 (Tag)	物理地址 47:12 位	物理地址 47:12 位	物理地址 47:12 位
写策略	不可写	写回法	写回法
读策略	非阻塞（2 个同时）	非阻塞 (24 个同时)	非阻塞（8 个同时）
读顺序	关键字优先	关键字优先	关键字优先
写顺序	不可写	顺序式	顺序式
校验手段	奇偶校验	ECC 校验	ECC 校验

[1] 物理地址具体落在哪个二级 Cache 模块 (4 个) 由交叉开关的路由配置决定。

5.2 一级指令 Cache

一级指令 Cache 大小是 64KB，采用四路组相联结构。Cache 行大小为 32 字节，可以存放 8 条指令。由于 GS464 的一级 Cache 采用 128 位的读通路，所以每个时钟周期可以取四条指令到超标量调度单元。

一级指令 Cache 实现了奇偶校验。当读一级指令 Cache 发生了奇偶校验错时，硬件会自动将相应的 Cache 行会被置成无效，并从二级 Cache 中获取正确的值。整个过程无需软件干预。

5.2.1 指令 Cache 的组织

图 5.1 给出了一级指令 Cache 的组织结构。该 Cache 采用四路组相联的映射方式，其中每组包括 512 个索引项。根据索引 (Index) 选择相应的标志域 (Tag) 和数据 (Data)。Cache 从读出 Tag 后，它被用来和虚地址中的被转换的部分进行比较，从而确定包含正确数据的组。

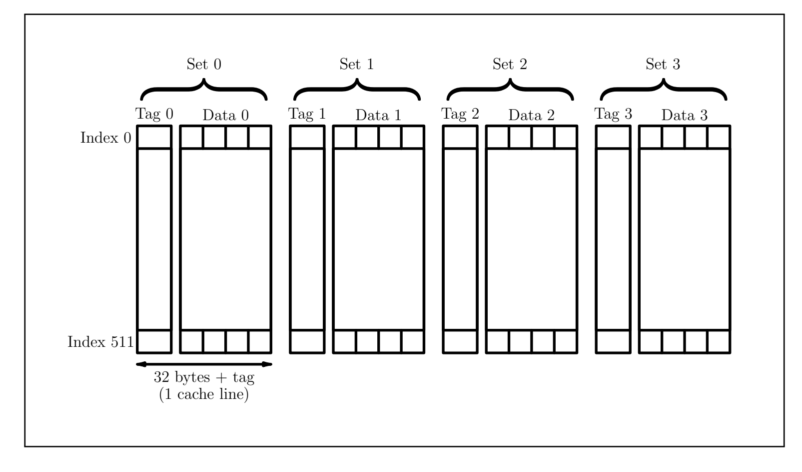


图 5.1: 指令 Cache 的组织结构

当一级指令 Cache 被索引时，四个组都会返回它们相应的 Cache 行，Cache 行大小为 32 字节，每个 Cache 行的标志域为 34 位，还有 1 位作为有效位。

5.2.2 指令 Cache 的访问

龙芯 GS464 指令 Cache 采用虚地址索引和物理地址标志的四路组相联结构。如图 5.2 所示，输入地址的低 14 位被用作指令 Cache 的索引。其中 13:5 位用于索引 512 个项。其中每个项中又包含四个 64 位的双字，4:3 位用来在这四个双字中进行选择。

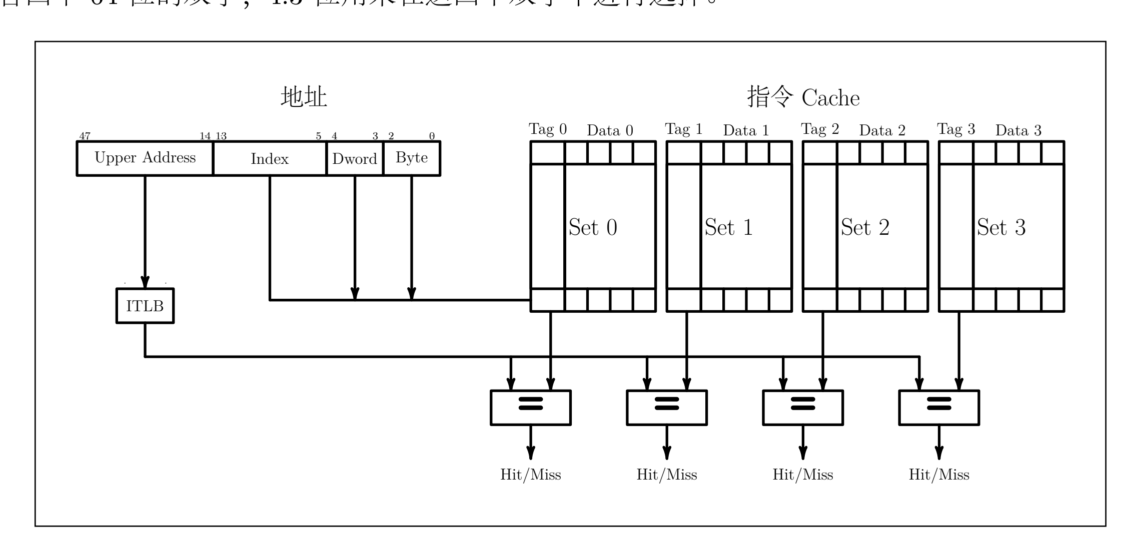


图 5.2: 指令 Cache 访问

当对 Cache 索引时，Cache 中取出四个块中的 Data 和相应的物理地址 Tag，从同时，高位地址通过指令 TLB 进行转换，将转换后的地址与取出的四个组中的 Tags 进行比较，若存在一个 Tag 与其匹配，则使用该组中的数据：这就被称为一次“一级 Cache 命中 (Hit)”；若四组的 Tag 都不与其匹配，那么中止操作，并开始访问二级 Cache：这就被称为“一级 Cache 脱靶 (miss)”。

5.3 一级数据 Cache

数据 Cache 的容量为 64KB，采用四路组相联的结构。Cache 块大小为 32 字节，即可以存放 8 个字。数据 Cache 的读写数据通路都是 128 位。

数据 Cache 使用的是虚地址索引，物理地址标志。操作系统可以解决可能由虚地址引起的页着色一致性问题。数据 Cache 是非阻塞的，也就意味着，数据 Cache 中的一次脱靶不会引起流水线的停顿。

数据 Cache 采用的写策略是写回法，即写数据到一级 Cache 的操作不会引起二级 Cache 和主存的更新。写回策略减少了一级 Cache 到二级 Cache 的通信量，从而提高了全局性能。只有在数据 Cache 行被替换出去时，数据才会被写到二级 Cache 中。

一级数据 Cache 实现了 ECC 校验。当读一级数据 Cache 发生了一位 ECC 校验错时，硬件会自动校正 Cache 读出的结果并将 Cache 中的内容更新为校正后的值。整个过程无需软件干预。当读一级数据 Cache 发生了二位 ECC 校验错时，将会发生例外留待软件处理。

5.3.1 一级数据 Cache 的组织

图 5.3 给出了数据 Cache 的组织结构。这是一个四路组相联的 Cache，其中含有 512 个索引项。当对 Cache 索引时，同时访问四个组中的 Tag 和 Data。然后将四个组中的 Tag 与转换后的物理地址部分进行比较，从而确定命中哪一个数据行。

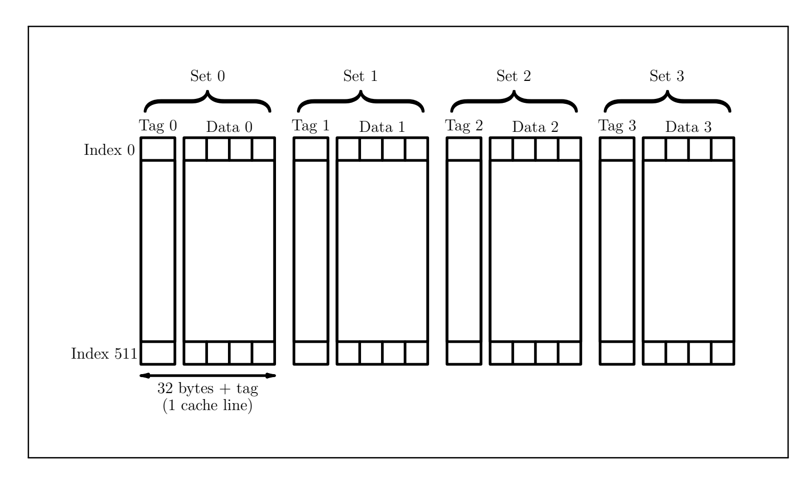


图 5.3: 一级数据 Cache 的组织结构

当索引数据 Cache 时，四个组中都会返回它们各自相应的 Cache 行。Cache 块大小为 32 字节，使用了 34 位作为物理标志位，1 位脏位还有 2 位状态位。Cache 行状态包括 INV，SHD 和 EXC 三种状态。其中，INV 状态表示该 Cache 行无效；SHD 状态表示该 Cache 行可读；EXC 状态表示该 Cache 行可读可写。

5.3.2 数据 Cache 的访问

龙芯 GS464 数据 Cache 采用虚地址索引和物理地址标志的四路组相联结构。图 5.4 给出了访问一次数据 Cache 时，虚地址的分解示意图。

如图所示，地址的低 14 位用作对数据 Cache 的索引。其中 13:5 位用作索引 512 个项，其中每个项又包括 4 个 64 位的双字。使用 4:3 位对四个双字进行选择，2:0 位用作选择一个双字的八个字节中的某一个字节。

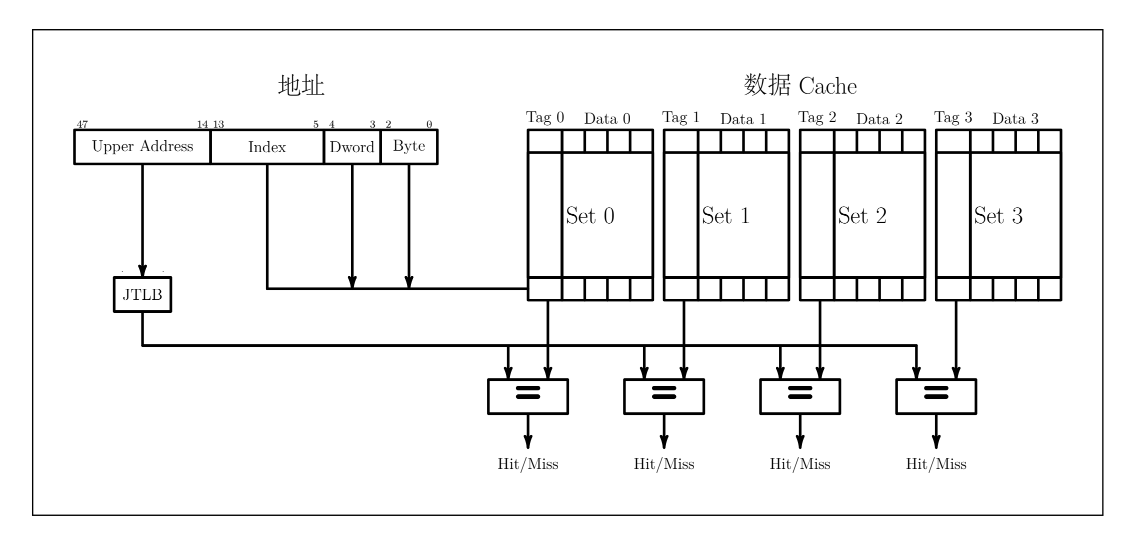


图 5.4: 数据 Cache 访问

数据 Cache 访问脱靶的指令（访存指令不命中或写指令命中 SHD 状态的 Cache 行），则访问二级 Cache。如果二级 Cache 命中，则将从二级 Cache 取回的 Cache 块送回一级 Cache。如果二级 Cache 继续脱靶，则将访问内存，从中取值填充二级 Cache 和数据 Cache。

5.4 二级 Cache

龙芯 GS464 二级 Cache 模块的容量为 1MB。每个 Cache 行大小为 32 字节。二级 Cache 模块的主要特征包括：采用 128 位 AXI 接口，四路组相连，8 项 Cache 访问队列，关键字优先，接收读失效请求到返回数据最快 8 拍，通过目录支持 Cache 一致性协议，可用于片上多核结构（也可直接和单处理器 IP 对接），软 IP 级可配置二级 Cache 模块的大小（512KB/1MB），采用四路组相联结构，运行时可动态关闭，支持 ECC 校验，支持 DMA 一致性读写和预取读，支持按窗口锁二级 Cache，保证读数据返回原子性。

二级 Cache 还维护了每个 Cache 行的目录，以记录每个一级 Cache 中是否包含该 Cache 行的备份。二级 Cache 采用的写策略是写回法。写回策略减少了总线的通信量，从而提高了系统的全局性能。只有在二级 Cache 行被替换出去时，数据才会被写到内存中。

二级 Cache 实现了 ECC 校验。当读二级 Cache 发生了一位 ECC 校验错时，硬件会自动校正 Cache 读出的结果并将 Cache 中的内容更新为校正后的值。整个过程无需软件干预。当读二级数据 Cache 发生了二位 ECC 校验错时，将会发生例外留待软件处理。

5.4.1 二级 Cache 的组织

二级 Cache 是混合 Cache，其中既包括指令也包括数据。二级 Cache 模块支持 Cache 一致性协议。根据 Cache 一致性的要求，二级 Cache 具有两方面的角色：对一级 Cache 来说是 home，对于内存来说是 Cache。当访问二级 Cache 时，同时访问四组的 Data 和 Tag，将取出的四个 Tag 分别和访问的物理地址高位部分进行比较，来确定数据是否还驻留在 Cache 中。

每个 Cache 行包含一个 32 字节的数据，31 位物理地址标志位，1 位 Cache 状态位（表示相应的 Cache 行在二级 Cache 中是否有效），1 位目录状态位（表示相应的 Cache 块是否在某个一级 Cache 中处于独占或共享状态）和 1 位 W 位（表示该行是否被写过）。

5.4.2 二级 Cache 的访问

只有在一级 Cache 脱靶的情况下，二级 Cache 才会被访问。二级 Cache 的索引和标志都采用的是物理地址。如图 5.5 所示，低位地址用来索引二级 Cache。四个组中都会返回它们各自相应的 Cache 行：17:5 位被用作二级 Cache 的索引；每个被索引项都含有 4 个 64 位的双字数据，使用 4:3 位在 4 个双字中进行选择；2:0 位用于选择一个双字中的某 8 个字节。

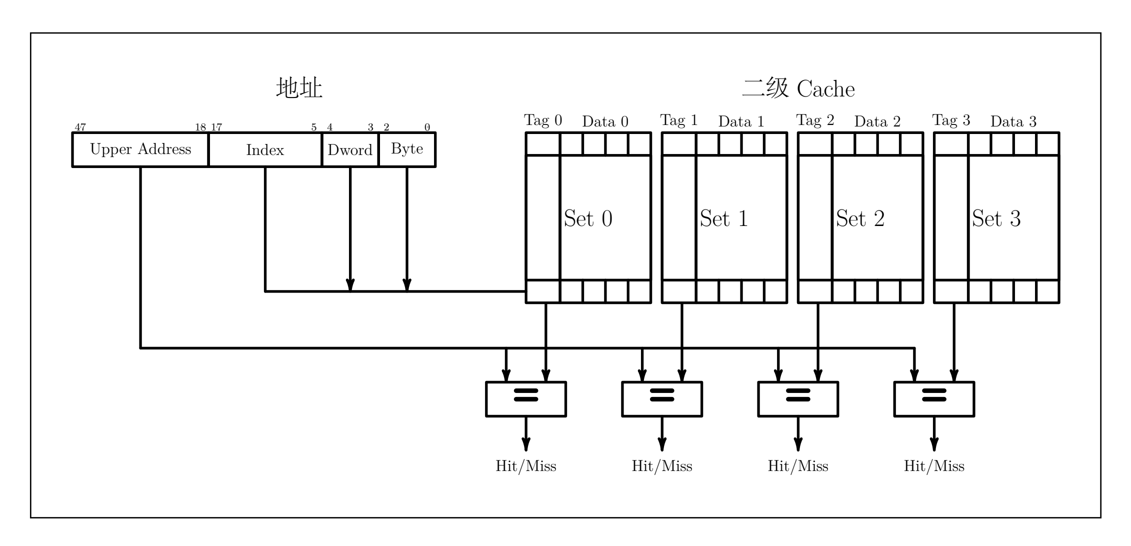


图 5.5: 二级 Cache 访问

5.5 Cache 一致性属性

龙芯 GS464 实现了表 5.2 所示的 Cache 算法和 Cache 一致性属性。

C(5:3)	Cache 一致性属性
0	— 保留 —
1	— 保留 —
2	非缓存（Uncached）
3	一致性缓存（Cacheable coherent）
4	— 保留 —
5	— 保留 —
6	— 保留 —
7	非缓存加速（Uncached Accelerated）

表 5.2: Cache 一致性属性含义

5.5.1 缓存一致性代码 2：非缓存（Uncached）

如果某个页采用非缓存算法时，那么对于在该页中任何位置的 Load 或 Store 操作，处理器都直接发射一个双字，部分双字，字，部分字的读或写请求给主存，而不通过任何一级 Cache。非缓存算法采用阻塞的方式实现。

5.5.2 缓存一致性代码 3：一致性缓存（Cacheable coherent）

一个具有该属性的行可以驻留在 Cache 中，相应的存数和取数操作都只访问一级 Cache。当一级 Cache 脱靶时，处理器会检查二级 Cache，看是否有包含所请求的地址。如果二级 Cache 命中，则从二级 Cache 中填充数据。如果二级 Cache 不命中，则从主存中取出数据，并将其写入二级 Cache 和一级 Cache。

由于存在多个处理器核及 IO 设备可以访问主存，因此龙芯 GS464 硬件实现了 Cache 一致性协议，无需通过软件采用 Cache 指令来主动维护 Cache 的一致性。

5.5.3 缓存一致性代码 7：非缓存加速（Uncached Accelerated）

非缓存加速属性用于优化在一个连续的地址空间中完成的一系列顺序的同一类型的不通过缓存的存数操作，它适用于对显示设备这类存储设备的快速输出访问。

该优化方法是通过设置缓冲区来收集这种属性的存数操作，只要缓冲区不满，就可以把这些存数操作的数据存入缓冲区中。缓冲区和一个 Cache 行一样大小，把数据存储到缓冲区中就和存储到 Cache 中一样。缓冲区满时，则进行块写操作。在顺序存数指令的收集过程中，若有其他类型的 Uncached 存数指令插入（比如随机存取指令），则收集工作中止，缓冲区中保存的数据按字节写方式输出。

Remark: 缓冲区在什么地方，内存?

第六章　处理器例外

本章详细介绍了龙芯 GS464 处理器核的例外（Exception）和例外处理。主要内容包括：例外的产生及返回，例外向量位置及所有支持的例外类型。针对每一种例外类型，还介绍了例外的原因，处理和服务。

6.1　例外概述

6.1.1　例外的产生及返回

当处理器开始处理某个例外时，CP0 Status 寄存器的 EXL 位将被置 1，这也意味着系统缺省运行在内核模式。同时 EPC 寄存器保存了引发例外的指令或例外发生是的指令的虚地址；如果该指令在分支延迟槽内, EPC 寄存器则保存了之前的分支指令的地址，并且将 Cause 寄存器的 BD 位置 1。在保存了适当的现场状态之后，例外处理程序通常将状态寄存器的 KSU 字段设定为内核模式，同时将 EXL 位置回为 0。当恢复现场状态并且重新执行被例外打断的程序时，处理程序则会把 KSU 字段恢复回上次的值，从例外程序返回会将 EXL 位置为 0。

6.1.2　例外优先级

当多个例外同时发生时，只有优先级最高的例外才会向处理器报告。表 6.1 列出了龙芯 GS464 支持的所有例外以及它们的优先级。

表 6.1: 例外优先级

例外优先顺序
冷重置 (最高优先级)
软重置
不可屏蔽中断 (NMI)
地址错误取指
TLB 重填取指
TLB 无效取指
Cache 错取指
总线错误取指
整型溢出，陷阱，系统调用，断点，保留指令，协处理器不可用，浮点例外
EJTAG 中断
地址错误数据存取

未完待续

表 6.1: 例外优先级（续）

例外优先顺序
TLB 重填数据存取
TLB 无效数据存取
TLB 修改写数据
EJTAG 数据断点
Cache 错数据存取
总线错误数据存取
中断 (最低优先级)

6.1.3 例外向量位置

一般而言，例外向量地址都是基地址加上向量偏移的形式。当状态寄存器中的 BEV （Boot-entry vector）位为 0 时，用户可通过 EBase 寄存器定义例外向量的基址。同时，几个重要的例外有它们自己的基地址规则：(1) 冷重置、软重置和非屏蔽中断 (NMI) 例外使用专用的重置例外向量地址 0xFFFF_FFFF_BFC0_0000，这个地址不通过 Cache 存取，也无需地址映射；(2) EJTAG 调试中断的入口根据其控制寄存器中的 ProbeTrap 位是 0 还是 1 分别选用 0xFFFF_FFFF_BFC0_0480 或 0xFFFF_FFFF_FF20_0200。表 6.2 列出了在各种情况下，各个例外对应的向量基址。

例外	条件	向量基地址
Reset, Soft Reset, NMI	—	0xFFFF_FFFF_BFC0_0000
EJTAG Debug	ProbEn=0	0xFFFF_FFFF_BFC0_0480
	ProbEn=1	0xFFFF_FFFF_FF20_0200
Cache Error	BEV=0	0xFFFFFFFF || EBase31..30 || 1 || EBase28..12 || 0x000
	BEV=1	0xFFFFFFFF BFC00300
通用例外	BEV=0	0xFFFFFFFF || EBase31..12 || 0x000
	BEV=1	0xFFFFFFFF BFC00200

表 6.2: 例外向量基地址

表 6.3 列出了各个例外向量相对于各自基地址的偏移。由于龙芯 GS464 处理器核提供了对向量中断模式 (Vectored Interrupt) 的支持，该模式由 Cause 寄存器的 IV 位选择。在传统模式下，外部中断 (包括时钟和性能计数器中断)，将使用通用例外入口及偏移，由软件负责分发到相应的服务。在向量中断模式下，中断优先级从 IP[7] 到 IP[0] 优先级依次降低并且有专门的例外入口。IntCtl 寄存器的 VS 域控制这些例外处理代码所占用的空间大小，每个中断对应的入口偏移可用下式计算 (其中向量号从零开始)：

```
向量中断偏移 = 0x200 + 向量号 * IntCtl[VS]
```

例外向量偏移表显示，TLB 和 XTLB 重填例外在 EXL=0 时有单独对应的例外偏移，这是为了加快对该类例外的处理。同样，TLB 和 XTLB 重填例外也可能在另外一个重填例外中产生，这种情况是有意义的，应该被允许。不过在这种情况下（系统通过 EXL=1 判断)，重填例外将使用通用例外入口。在以下对 TLB/XTLB 重填例外的介绍中，我们还会进一步阐述这个问题。

例外	例外向量偏移
Reset, Soft Reset, NMI	0x000
TLB Refill, EXL=0	0x000
XTLB Refill, EXL=0	0x080
Cache error	0x100
通用例外	0x180
向量中断（Cause[IV]=1）	0x200

表 6.3: 例外向量偏移

以下将按照例外优先级的顺序依次介绍各个例外。为了方便，某些特定例外，如 TLB 例外和指令/数据例外等，将放在一起介绍。在以下各节介绍中，每个例外都将先由硬件来处理，然后由软件来服务。

6.2 详细例外说明

6.2.1 冷重置例外

原因 当系统第一次上电或者冷重置时，将产生冷重置例外。该例外不可屏蔽。

处理 CPU 为这个例外提供了一个专门的中断向量供其使用（32 位模式下为 0xBFC0_0000，64 位模式下为 0xFFFF_FFFF_BFC0_0000）。该地址是无需地址映射和不通过 Cache 存取数据的 CPU 地址空间。因为该例外需要在 Cache 和 TLB 都处于不确定状态时，处理器也可以正常取指并执行指令。

当例外发生时，除下列寄存器域外，CPU 中所有寄存器内容是不确定的：

- Status 寄存器的初值为 0x30C0_00E4: SR 位被清 0，ERL 和 BEV 位被置 1；
- 配置（Config）寄存器的初值为 0x8003_4482；
- 随机（Random）寄存器初始化为它的最大值；
- Wired 寄存器初始化为 0；
- ErroEPC 寄存器初始化为 PC 的值；
- PerfCnt 寄存器的 Event 位初始化为 0；
- 所有断点和外部中断都被清除。

服务 冷重置例外的服务包括：初始化所有的处理器寄存器，协处理器，Cache 和存储系统；执行诊断测试；引导并自举操作系统。

6.2.2 NMI 例外

原因 当信号 NMIn 值为低时，将产生 NMI 例外。该例外不可屏蔽。

处理 发生 NMI 例外时，状态寄存器的 SR 位被置为 1，用以区分冷重置。NMI 例外只能在指令的边界被提取。它并不抛弃任何机器的状态，而是保留处理器的状态用于诊断，系统直接跳转到 NMI 例外处理程序开始处开始执行。

NMI 例外保留了除下列寄存器/域外的所有寄存器值:

- 包含 PC 值的 ErrorEPC 寄存器。
- 置为 1 的状态寄存器 ERL 位。
- 软重置或 NMI 置为 1，冷重置置为 0 的状态寄存器 SR 位。
- 置为 1 的状态寄存器 BEV 位;
- PC 寄存器重置为 0xFFFF_FFFF_BFC0_0000。

服务 NMI 例外可于除了 “重置处理器，同时保持 Cache 和内存内容” 之外的所有各种需要重置处理器的情形例如，当检测到电源故障时，系统可以通过 NMI 例外立即可控地关闭系统。由于 NMI 例外通常在另外一个错误例外中发生，因此从例外返回后，通常不太可能继续执行原来的程序。

6.2.3 地址错误例外

原因 当以下情况发生时，将发生地址错误例外:

- 引用非法地址空间:
  - 在用户模式下引用超级用户地址空间;
  - 在用户或超级用户模式下引用内核地址空间。
- 存取边界未对齐:
  - 取存一个双字，但双字未对齐于双字边界;
  - 取存一个字，但字未对齐于字的边界;
  - 取或存一个半字，但半字未对齐于半字的边界。

该例外不可屏蔽。

处理 该例外使用通用例外向量入口，并且 Cause 寄存器的 ExcCode 字段编码设为 AdEL 或 AdES；BadVAddr 寄存器保存了没有正确对齐的虚地址，或者受保护的地址空间的虚地址；如果引发例外的指令不在分支延迟槽中，那么 EPC 寄存器保存了该指令的地址，否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。这些信息可以确认引发例外的指令以及例外起因是指令引用、取操作指令还是存操作指令。

服务 导致例外发生的进程会收到 UNIX SIGSEGV(段违例) 信号，这个错误对一般进程来说通常是致命的。

6.2.4 TLB 重填例外

原因 当 TLB 中没有项匹配映射地址空间的引用地址时，TLB 重填例外发生，该例外是不可屏蔽的。

处理 针对这个例外，MIPS 体系结构提供了两个专门的例外向量，分别用于 32 位和 64 位地址空间情形。它们有不同的地址偏移：当引用地址为 32 位地址空间模式时，例外向量偏移为 0x000；当引用地址为 64 位地址空间模式时，偏移为 0x080。

当状态寄存器 EXL 位为 0 时，TLB 重填例外将使用专门例外向量入口。这个例外设置 Cause 寄存器中 ExcCode 字段的编码值为 TLBL 或 TLBS。如果引发例外的指令不在分支延迟槽中，那么 EPC 寄存器保存了导致例外的指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置 1。ExcCode 编码与 EPC 寄存器以及 BD 位一起，指明了引起例外的指令以及例外的起因是指令引用、取操作指令还是存操作指令。发生这个例外时，BadVAddr 和 EntryHi 寄存器保存了地址转换失败的指令虚地址。Context 或 XContext 寄存器含有指向页表的地址指针。EntryHi 寄存器还保存了转换失败时的 ASID。Random 寄存器通常保存了用于放置被替换 TLB 项的合法位置。EntryLo 寄存器的内容是不确定的。

当状态寄存器 EXL 位值为 1 时，即 TLB 重填例外在另一个重填例外中发生，这种情况下，TLB 重填例外将使用通用例外向量入口，而且处理器保持 EPC 寄存器值，这样处理结束后，控制将回到最初的程序继续执行。

服务 例外服务将通过 Context 或 XContext 指向的页表取得引发例外的虚地址对应缺失页信息。这些信息包括两个相邻页面的 TLB 项信息及其访问控制位信息。这一对页表信息将被写入 EntryLo0/EntryLo1 寄存器，然后被写入 TLB 表项。

用于获得页面信息和访问控制信息的虚地址也有可能位于一个没有驻留在 TLB 中的页面上。如果出现这种情况，则 TLB 重填处理程序将引发另外一个 TLB 重填例外。由于 Status 寄存器的 EXL 位为 1，第二个 TLB 重填例外将使用通用例外向量入口[1]。

6.2.5 TLB 无效例外

原因 当一个虚地址引用匹配到一项被标记为无效的 TLB 项 (TLB 有效位被清掉) 时，TLB 无效例外发生。当下面情况之一发生时，TLB 项将被标记为无效：

- 虚地址不存在；
- 虚地址存在，但是不在内存中（缺页）；
- 利用这个特性维护一个页面引用位：任何对该页面的引用将引发一个陷阱。

这个例外是不可屏蔽的。

[1] 严格说来，第二个重填例外劫持了第一个重填例外，它完成后将直接返回到触发例外的进程 (因为 EPC 未被修改)。不过，另一个重填例外会马上再次被触发因为缺失的页面依然缺失，不过这一次因为缺失的页表页面信息已经存在 TLB 中，所以能够顺利地完成

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 TLBL 或 TLBS。如果引发例外的指令不在分支延迟槽内，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置 1。BadVAddr 和 EntryHi 寄存器保存了地址转换失败的虚地址，EntryHi 寄存器还保存了转换失败时的 ASID。Random 寄存器则包含有用于替换的 TLB 表项位置。EntryLo 寄存器的内容为不确定的。

服务 在服务完 TLB 无效例外的起因后，TLBP 指令将用来定位该无效的 TLB 项，然后用标记有效的一项来替换。

6.2.6 TLB 修改例外

原因 当写内存操作的虚地址引用与 TLB 中某项匹配，但该项并没有被标示为“脏”，即不可写时，TLB 修改例外发生。该例外不可屏蔽。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器中的 ExcCode 字段编码为 MOD。如果引发例外的指令不在分支延迟槽内，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。BadVAddr 和 EntryHi 寄存器保存了地址转换失败的虚地址，EntryHi 寄存器还保存了转换失败时的 ASID。EntryLo 寄存器的内容是不确定的。

服务 内核将使用失败的虚地址和虚页号来再次确认页面的访问控制信息。如果页面的写访问是不允许的，那么写保护违例发生。该错误对一般进程来说是致命的。如果写访问是允许的，那么内核将在页表结构内将页标记为可写，先更新 EntryLo 寄存器，然后用 EntryHi 和 EntryLo 寄存器更新对应的 TLB 表项。

6.2.7 Cache 错误例外

原因 当处理器取指或者访存而出现内部 Cache 校验错时，Cache 错误例外发生。该例外不可屏蔽。

处理 该例外使用偏移量为 0x100 的专用 Cache 错例外入口。该例外入口在不经过 Cache 就可以访问的地址段。例外发生时，Cause 寄存器的 ExcCode 字段编码为 CacheErr。如果引发例外的指令不在分支延迟槽内，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。这些信息可以确认引发例外的指令以及例外的起因是指令引用还是访存操作指令等。同时，CacheErr 寄存器记录了出错类型和在组相联 Cache 中的位置，CacheErr1 寄存器记录出错的指令虚地址或者内存物理地址。

服务 龙芯 GS464 处理器核对 Cache 错实现了硬件自纠错功能：若为指令 Cache 出错，整个出错的 Cache 行将被替换；若为数据 Cache 出错且只有一位错时，错误数据将被自动纠正；若数据 Cache 出错且有两位错时，则需要操作系统的介入，根据出错数据块的位置等信息决定如何处理。

6.2.8 总线错误例外

原因 当处理器进行数据块读取更新，或双字/单字/半字的读请求时收到外部的 ERR 完成应答信号，则总线错误例外发生。该例外不可屏蔽。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 IBE 或 DBE。如果引发例外的指令不在分支延迟槽内，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。这些信息可用于确认引发例外的指令，以及例外起因是指令引用、取操作指令还是存操作指令。

服务 例外服务程序可以通过 CP0 寄存器中的信息计算出引发错误的物理地址。EPC 寄存器保存导致例外发生的指令虚地址（如果 Cause 寄存器的 BD 位被置为 1，则该指令的虚地址为 EPC 寄存器内容加 4)。如果 Cause 寄存器中的 ExcCode 字段编码为 IBE (取指引用出错)，那么该指令则是引发例外的原因。如果 Cause 寄存器中的 ExcCode 字段值编码为 DBE (读取或存储引用出错)，则需要通过解释这条指令来获得引发例外的读取和存储引用的虚地址。对应的物理地址可以通过 TLBP 指令以及读取 EntryLo 寄存器包含的物理页号来获得。

引发例外的进程将收到 UNIX SIGBUS（总线错误）信号，对一般进程来说这通常是致命的。

6.2.9 整型溢出例外

原因 当执行 ADD、ADDI、SUB、DADD、DADDI 或 DSUB 指令，结果补码溢出时，整型溢出例外发生。该例外不可屏蔽。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 OV。如果引发例外的指令不在分支延迟槽内，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置 1。

服务 引发例外的进程会收到 UNIX SIGFPE/FPE_INTOVE_TRAP （浮点例外/整型溢出）信号。对一般进程来说，这个错误通常是致命的。

6.2.10 陷阱例外

原因 当指令 TGE、TGUE、TLT、TLTU、TEQ、TNE、TGEI、TGEUI、TLTI、TLTUI、TEQI、TNEI 执行条件结果为真时，陷阱例外发生。该例外不可屏蔽。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 TR。如果引发例外的指令不在分支延迟槽内，那么 EPC 寄存器保存了该指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且 Cause 寄存器的 BD 位被置为 1。

服务 引发例外的进程会收到一个 UNIX SIGFPE/FPE_INTOVE_TRAP （浮点例外/整型溢出）信号。对一般进程来说，这个错误通常是致命的。

6.2.11 系统调用例外

原因 当执行 SYSCALL 指令时，系统调用例外发生。该例外不可屏蔽。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 SYS。如果 SYSCALL 指令没有在分支延迟槽中，则 EPC 寄存器保存该指令的地址；否则，EPC 寄存器保存有之前的分支指令的地址，且将状态寄存器中的 BD 位被置 1。

服务 例外发生时，控制权将转到适当的系统调用例程。根据 SYSCALL 指令的 Code 字段 (位 25：6) 以及 EPC 寄存器中所存地址的指令，可以进一步分析具体系统调用的细节。当恢复进程的执行时，必须改变 EPC 寄存器的内容，以防止同一条 SYSCALL 指令再次被调用。这一般可以通过在返回前在 EPC 寄存器的值加 4 来完成；如果 SYSCALL 指令处在分支延迟槽中，则需要分析 EPC 中分支指令以取得跳转地址。

6.2.12 断点例外

原因 当执行 BREAK 指令时，断点例外发生。该例外不可屏蔽。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 BP。如果 BREAK 指令没有在分支延迟槽中，EPC 寄存器保存这条指令的地址；否则，EPC 保存之前的分支指令的地址，并且将状态寄存器的 BD 位置 1。

服务 例外发生时，控制权被转到适当的系统例程。进一步的区分可以分析 BREAK 指令的 Code 字段（位 25：6），以及载入 EPC 寄存器中所存地址的指令的内容。如果这条指令在分支延迟槽中，那么 EPC 寄存器中的内容必须加上 4 以定位到该指令。为了恢复进程的执行，必须改变 EPC 寄存器的内容，这样 BREAK 指令才不会再次被执行；这可以通过在返回之前使 EPC 寄存器的值加 4 来完成。如果 BREAK 指令在分支延迟槽中，那么为了恢复进程的继续执行，需要解释该分支指令。

6.2.13 保留指令例外

原因 当试图执行一条没有在 MIPS64 R2 定义，亦非龙芯自定义指令时，保留指令例外发生。该例外不可屏蔽。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 RI。如果引发例外的指令没有在分支延迟槽中，EPC 寄存器保存这条指令的地址；否则，保存之前的分支指令地址，并且将状态寄存器的 BD 位置 1。

服务 导致例外发生的进程会收到 UNIX SIGILL/ILL_RESOP_FAULT （非法指令/保留的操作错误）信号。对一般进程来说，这个错误通常是致命的。

6.2.14 协处理器不可用例外

原因 以下任何一种情况就触发协处理器不可用例外:

- 进程在非内核模式下试图执行 CP0 指令，而 CP0 单元没有被标记为可用;
- 指令对应的协处理器单元（CP1 或 CP2）没有被标记为可用;

该例外不可屏蔽。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 CpU，CE 域指示四个协处理器的哪个被引用。如果这条指令不在分支延迟槽中，EPC 寄存器保存了不可使用协处理器指令的地址；否则，EPC 寄存器保存了之前的分支指令的地址，并且状态寄存器的 BD 位被置 1。

服务 如果进程被授权访问协处理器，则协处理器被标记为可用，相应的用户状态被恢复以执行协处理器指令。如果进程有授权，但是协处理器不存在或者有故障，则需要解释/模拟该条协处理器指令。如果该指令在分支延迟槽中，则必须解释执行分支指令，然后模拟执行协处理器指令。

如果进程没有被授权访问协处理器，引发例外的进程会收到 UNIX SIGILL/ILL_PRIVIN_FAULT （非法指令/特权指令错误）信号。该错误通常是致命的。

6.2.15 浮点例外

原因 浮点协处理器使用浮点例外。这个例外是不可屏蔽的。

处理 该例外使用通用例外向量入口。例外发生时，Cause 寄存器的 ExcCode 字段编码为 FPE。浮点控制/状态 FCSR 寄存器存有这个浮点例外产生的原因等信息。

服务 清除浮点/状态寄存器中的适当位可以清除这个例外。

6.2.16 EJTAG 例外

当与某些 EJTAG 相关的条件满足时，触发 EJTAG 例外。具体描述见第 X 章

6.2.17 中断例外

原因 当八个中断条件（分别对应 Cause 寄存器的 IP 域的八位）中的任何一个被触发时，中断例外发生。这些中断的设定和优先级依赖于具体的系统实现。通过清掉在状态寄存器中的 IM 域中的相应的位，可以屏蔽任何一个特定的中断；清掉状态寄存器的 IE 位，可以屏蔽所有的八个中断。

处理 例外发生时，Cause 寄存器的 ExcCode 字段编码为 INT。根据 Cause 寄存器的 IV 位配置，处理器将决定使用传统的通用例外向量入口，或者使用向量化例外模式，选择最高优先级的中断号对应的入口进行处理。Cause 寄存器中的 IP 域指明了当前的中断请求。不止一个的中断位可能同时被设置；如果中断触发并且在寄存器被读到之前被撤消，甚至没有位被设置。时钟中断和性能计数器溢出中断由 Cause 寄存器中的 TI 和 PCI 位指示：这两个中断都由 IP[7] 位控制。

如果未用向量中断模式，因为一个中断同时可能有多个源，软件需要对每一个可能的中断源进行查询来判断中断产生的原因。

服务 如果中断是由两个软件产生例外之一导致的，则设置 Cause 寄存器中的相应位，IP[1:0] 为 0 可以清除中断条件。定时器中断的清除通过写 Compare 寄存器来完成。性能计数器中断的清除则是向计数器的溢出位，即位 31，写入 0 来实现。如果中断是硬件产生的，那么撤消引起触发的中断管脚的条件，就可清除中断条件。

注意，软件中断是非精确的：一旦软件中断触发，在例外被处理之前，程序还可能继续执行。冷重置和软重置会清除所有未完成的外部中断请求，IP[2] 至 IP[6]。

第七章　浮点协处理器

浮点协处理器（Floting Point Unit，简称 FPU）是一个重要的 CPU 协处理器，一般也被称为 CP1 （Coprocessor 1）：它通过扩展 CPU 的指令集来完成浮点算术运算功能。本章主要讲述了龙芯 GS464 处理器的 FPU 的特性，主要内容包括编程模型、指令集、指令格式、指令流水线以及异常等。在构架层面，龙芯 GS464 的浮点协处理器及其相关的系统软件实现符合 ANSI/IEEE 754–1985 的二进制浮点运算标准。同时，龙芯 GS464 浮点协处理器还拥有自定义的 SIMD 多媒体定点 (fixed-point) 指令集。

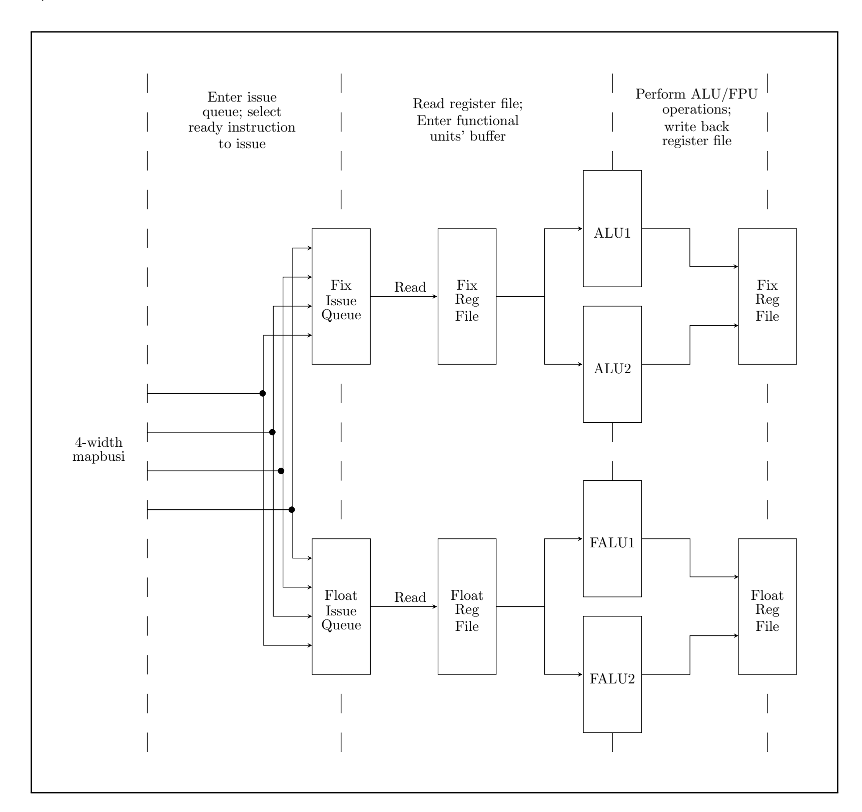


图 7.1: 龙芯 GS464 浮点功能单元的组织构成

龙芯 GS464 的 FPU 由 FALU1 和 FALU2 两个功能单元组成。图 7.1 对 GS464 的浮点功能单元的组织构成进行了图解说明。

- FALU1 单元: FALU1 模块可以执行除浮点访存以及浮点和定点数据传送之外的所有浮点操作, 包括浮点加（减）法、浮点乘法、浮点乘加（减），浮点除法，浮点开平方根，浮点求倒，浮点开根后求倒，浮点与定点转换, 浮点精度转换, 浮点比较, 转移判断和其它简单逻辑等。此外，FALU1 模块通过指令编码中 FMT 域的扩展与复用来执行 SIMD 媒体操作。
- FALU2 单元: FALU2 执行浮点乘加运算部件（可计算浮点乘、加和浮点乘加指令），以及

多媒体指令操作。同时，龙芯 GS464 的 FPU 支持执行 MIPS64 R2 指令集中的单精度对（Paired-Single，简称 PS）浮点指令。

龙芯的浮点队列每个时钟周期可以各分别发射 1 条指令到 FALU1 和 FALU2。浮点寄存器文件为这两个单元各提供三个专用的读端口和一个专用的写端口。

7.1 浮点寄存器

这部分主要讲述 GS464 的浮点寄存器组及其组织结构。浮点寄存器组又可分为浮点控制寄存器（简称 FCR）和浮点通用寄存器（简称 FGR）。总体而言，龙芯 GS464 的浮点寄存器与 MIPS64 R2 的浮点寄存器兼容，但略有差别。

在浮点通用寄存器上，龙芯 GS464 的浮点寄存器设置沿袭了 R10000 用法，与 MIPS64 略有不同。这具体体现在当 CP0 的 Status 寄存器的 FR 位为 1 时，表明系统有 32 个 64 位的逻辑浮点通用寄存器可用；而当 FR 位为 0 时，表明只有 16 个 32 位或 64 位的逻辑浮点寄存器。与之相对的是，在 MIPS64 的设定上，FR 的值为 1 表示有 32 个 32 位的逻辑浮点通用寄存器，而 FR 为 0 表示有 16 个 64 位的逻辑浮点通用寄存器。

在物理实现上，GS464 的每个浮点单元有 64 个物理浮点通用寄存器。32 个逻辑浮点通用寄存器通过重命名模块后映射到这些物理寄存器上。

在浮点控制寄存器上，MIPS64 R2 规定每个协处理器最多可以拥有 32 个控制寄存器。而龙芯 GS464 核的浮点控制寄存器实现了 5 个控制寄存器，它们分别是：FIR（1 号）、FCSR（31 号）、FCCR（25 号）、FEXR（26 号）和 FENR （28 号）。

7.1.1 FIR 寄存器（FCR0）

FIR （FPU Implementation and Revision 寄存器）是 32 位的只读寄存器。它包含了浮点单元实现的功能、处理器 ID、修订版本号等诸多信息。GS464 里 FIR 值为 0x0077_0501。表 7.1 显示了 FIR 寄存器的格式及其寄存器各域的含义。

表 7.1: FIR 寄存器

31–28	27–24	23	22	21	20	19	18	17	16	15–8	7–0
0	Impl	0	F64	L	W	3D	PS	D	S	ProcessorID	Revision
0 0 0 0	0 0 0 0	0	1	1	1	0	1	1	1	0 0 0 0 0 1 0 1	0 0 0 0 0 0 0 1
4	4	1	1	1	1	1	1	1	1	8	8

位域	描述
Impl	实现相关
F64	浮点数据通路是否为 64 位 0 － 32 位 1 － 64 位
L	长字（64 位）定点数据类型是否实现 0 －未实现 1 －已实现
W	字（32 位）定点数据类型是否实现 0 －未实现 1 －已实现
3D	MIPS-3D ASE 是否实现 0 －未实现 1 －已实现
PS	浮点对数据类型是否实现 0 －未实现 1 －已实现
D	双精度浮点数据类型是否实现 0 －未实现 1 －已实现
S	单精度浮点数据类型是否实现 0 －未实现 1 －已实现

未完待续

表 7.1: FIR 寄存器（续）

位域	描述
ProcessorID	浮点处理器标识
Revision	浮点单元的修订版本号
0	保留。必须按 0 写入，读时返回 0。

7.1.2 FCSR 寄存器（FCR31）

FCSR （FPU Control and Status Register）是一个 32 位的可读写寄存器，用于控制浮点单元的操作和显示浮点运算结果的状态。在 GS464 中 FCSR 的初始值为 0x00000F80。表 7.2 显示了 FCSR 寄存器的格式，及其各域含义。其中单字母助记符 E、V、Z、O、U、I 分别表示：未实现操作、无效操作、除零、上溢、下溢、不精确。

表 7.2: FCSR 寄存器

位	31–25	24	23	22	21–18	17–12	11–7	6–2	1–0
域	CC7-CC1	FS	CC0	T	0	Cause E V Z O U I	Enables V Z O U I	Flags V Z O U I	RM
宽度	7	1	1	1	4	6	5	5	2

位域	描述
CC7-0	浮点条件码域：记录浮点比较结果，用于条件跳转或转移等。
FS	冲刷位：该位设置时，非正常数的结果将被刷为 0，而不产生例外。
T	Top 模式位：该位为龙芯具体实现位，用于在译码时指示是否使用 x86 的 TOP 寄存器对浮点寄存器号进行重命名。
Cause	浮点例外原因域。
Enables	浮点例外使能域。
Flags	标志域：是否有 IEEE 的浮点例外产生。Enables 里未打开时，可查看本字段。
RM	舍入模式（rounding mode）域。
0	保留。必须按 0 写入，读时返回 0。

浮点条件域（CC7-0）

当一个浮点比较操作发生时，结果被保存在 CC0 条件位。如果比较结果为真，则 CC0 位被置 1；反之则置 0。CC0 位仅能被浮点比较指令和 CTC1 指令所修改。

类似的，其他条件位对应相应的浮点操作。CC1-0 位对应单精度对的条件比较结果。更多的细节参照 PS 和 SIMD 指令相关章节。

浮点例外原因域（Cause）

浮点例外 Cause 域是 CP0 的 Cause 寄存器的一个逻辑扩充：这些位指示了最近一次浮点操作所引起的例外情况。如果相应的浮点例外使能位（Enable）被设置的话，则将产生一个中断或者例外。如果一条指令可能引发多个例外，则每一个相应的例外原因位都会被设置。

Cause 域可能被除 Load、Store、Move 操作外的所以浮点操作指令所重写。在软件仿真实现某些指令的情形下，则需要将未实现操作位（E）置 1。其它位则可依照 IEEE754 标准予以设置。

注意：当一个浮点例外发生时，任何结果都**不会**被储存，所有的可用信息反应在 Cause 域上。

浮点例外使能域（Enables）

任何时候当 Cause 位和相应的使能位（Enable）同时为 1 时，则将产生一个浮点例外。如果浮点操作设置了一个被允许激活（相应使能位为 1）的 Cause 位，则处理器会立即产生一个例外，这和用 CTC1 指令同时设置 Cause 位和 Enable 位为 1 的效果一样。未实现操作（E）没有相应的使能位：当该位被设置时，浮点例外一定会发生。

从一个浮点例外服务程序返回之前，软件首先必须用 CTC1 指令来清除被激活了的 Cause 位以防止中断的重复执行。因此，如果某 Cause 位被使能时（对应的 Enable 位为 1），用户态程序将永远不会观察到该位值为 1：因为有更高优先权的例外服务程序会在用户程序之前服务例外，并清除该位。如果用户态程序需要该信息，则必须将其内容储存到其它地方而不是在 FCSR 寄存器中。如果该 Cause 位没有被使能，根据 IEEE754 标准，运算的结果将被写回，而且没有例外发生。在这种情况下，前一条浮点指令所引起的例外情形能够通过读 Causes 域的值来确定。

浮点标志域（Flags）

标志域的各标志位指示自从上次被重置后是否发生了对应的浮点例外。如果一个对应的浮点例外曾产生，那么相应的 Flag 位被置 1，否则保持不变。任何浮点运算都不会清除这些标志位，我们只能通过 CTC1 控制指令写一个新值到 FCSR 寄存器中来实现对 Flag 位的设置或清除。

注意：Flags 域不由硬件来设置；浮点例外服务程序需要在调用用户程序前设置这些位。

浮点舍入模式域（RM）

FCSR 寄存器的第 0 位和第 1 位组成了舍入模式（RM）域。所有的浮点运算根据这些位所指定的舍入方式进行相应的舍入处理。表 7.3 列出了所有的舍入模式编码值。

RM[1:0]	助记符	描述
00_2	RN	把结果向最接近可表示数的方向舍入，当两个最接近可表示数离结果一样接近时，则向最低位为 0 的那个最接近数方向舍入。
01_2	RZ	向 0 方向舍入：把结果向与之最接近并且在绝对值上不大于它的那个数舍入。
10_2	RP	向正无穷大方向舍入: 把结果向与之最接近并且不小于它的那个数舍入
11_2	RM	向负无穷大方向舍入：把结果向与之最接近并且不大于它的那个数舍入

表 7.3: 浮点舍入模式位（RM）编码

7.1.3 FCCR 寄存器（FCR25）

FCCR （FPU Condition Code Register）寄存器是访问浮点条件码域 CC7-0 的另一种方式。其内容与 FCSR 里的 CC 域完全相同，不同的是本寄存器中的 CC 域各位是连续的。表 7.4 显示了 FCCR 寄存器的格式。

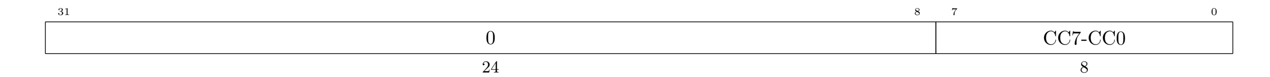


表 7.4: 浮点 FCCR 寄存器

7.1.4 FEXR 寄存器（FCR26）

FEXR 寄存器是访问 Cause 和 Flags 字段的另一种方式，其内容与 FCSR 里的相应字段完全相同。表 7.5 显示了 FEXR 寄存器的格式。

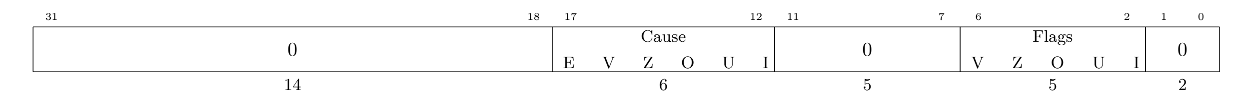


表 7.5: 浮点 FEXR 寄存器

7.1.5 FENR 寄存器（FCR28）

FENR 寄存器是访问 Enable, FS 和 RM 字段的另一种方式，其内容与 FCSR 里的相应字段完全相同。表 7.6 显示了 FENR 寄存器的格式。

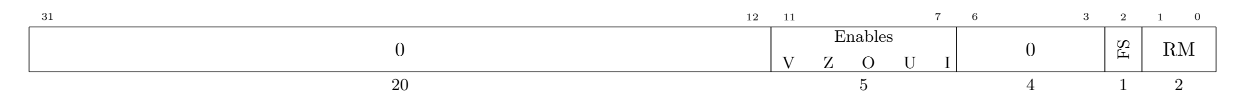


表 7.6: 浮点 FENR 寄存器

7.2 FPU 指令集概述

GS464 实现了 MIPS64 中浮点部分的所有数据类型，包括 S，D，W，L，和可选的 PS 类型。表 7.7 列出了 GS464 中 MIPS64 相关的所以浮点指令。

表 7.7: MIPS64 的浮点指令集

指令 OpCode	描述	MIPS ISA
算术指令		
ABS.*fmt*	绝对值	MIPS32
ADD.*fmt*	加法	MIPS32
DIV.*fmt*	除法	MIPS32
MADD.*fmt*	乘加	MIPS64
MSUB.*fmt*	乘减	MIPS64
MUL.*fmt*	乘法	MIPS32
NEG.*fmt*	求反	MIPS32
NMADD.*fmt*	乘加后求反	MIPS64
NMSUB.*fmt*	乘减后求反	MIPS64
RECIP.*fmt*	求倒数	MIPS64

未完待续

表 7.7: MIPS64 的浮点指令集（续）

指令 OpCode	描述	MIPS ISA
RSQRT.*fmt*	平方根后求倒数	MIPS64
SQRT.*fmt*	平方根	MIPS32
SUB.*fmt*	减法	MIPS32
分支跳转指令		
BC1F	浮点假时跳转	MIPS32
BC1FL	浮点假时 Likely 跳转	MIPS32
BC1T	浮点真时跳转	MIPS32
BC1TL	浮点真时 Likely 跳转	MIPS32
比较指令		
C.*cond*.*fmt*	比较浮点值并置标志位	MIPS32
转换指令		
ALNV.PS	可变浮点对齐	MIPS64
CEIL.L.*fmt*	浮点转换到 64 位定点，向上取整	MIPS64
CEIL.W.*fmt*	浮点转换到 32 位定点，向上取整	MIPS64
CVT.D.*fmt*	浮点或定点转换到双精度浮点	MIPS32
CVT.L.*fmt*	转换浮点值到 64 位定点	MIPS64
CVT.PS.S	转换两个浮点值到浮点对	MIPS64
CVT.S.PL	转换浮点对的低位到单精度浮点	MIPS64
CVT.S.PL	转换浮点对的高位到单精度浮点	MIPS64
CVT.S.*fmt*	浮点或定点转换到单精度浮点	MIPS32
CVT.W.*fmt*	转换浮点值到 32 位定点	MIPS32
FLOOR.L.*fmt*	浮点转换到 64 位定点，向下取整	MIPS64
FLOOR.W.*fmt*	浮点转换到 32 位定点，向下取整	MIPS64
PLL.PS	合并两个浮点对的低位为新的浮点对	MIPS64
PLU.PS	合并两个浮点对的低位和高位为新的浮点对	MIPS64
PUL.PS	合并两个浮点对的高位和低位为新的浮点对	MIPS64
PUU.PS	合并两个浮点对的高位为新的浮点对	MIPS64
ROUND.L.*fmt*	把浮点数四舍五入到 64 位定点	MIPS64
ROUND.W.*fmt*	把浮点数四舍五入到 32 位定点	MIPS32
TRUNC.L.*fmt*	把浮点数向绝对值小的方向舍入到 64 位定点	MIPS64
TRUNC.W.*fmt*	把浮点数向绝对值小的方向舍入到 32 位定点	MIPS32
访存指令		
LDC1	从内存取双字	MIPS32
LDXC1	按索引从内存取双字	MIPS64
LUXC1	按非对齐索引从内存取双字	MIPS64
LWC1	从内存取字	MIPS32
LWXC1	按索引从内存取字	MIPS64
SDC1	存双字到内存	MIPS32

未完待续

表 7.7: MIPS64 的浮点指令集（续）

指令 OpCode	描述	MIPS ISA
SDXC1	按索引存双字到内存	MIPS64
SUXC1	按非对齐索引存双字到内存	MIPS64
SWC1	存字到内存	MIPS32
SWXC1	按索引存字到内存	MIPS64
MOVE 指令		
CFC1	读浮点控制寄存器到 GPR	MIPS32
CTC1	写浮点控制寄存器到 GPR	MIPS32
DMFC1	从 FPR 复制双字到 GPR	MIPS64
DMTC1	从 GPR 复制双字到 FPR	MIPS64
MFC1	从 FPR 复制低字到 GPR	MIPS32
MFHC1	从 FPR 复制高字到 GPR	MIPS32 R2
MOV.*fmt*	复制 FPR	MIPS32
MOVF.*fmt*	浮点假时复制 FPR	MIPS32
MOVN.*fmt*	GPR 不为 0 时复制 FPR	MIPS32
MOVT.*fmt*	浮点真时复制 FPR	MIPS32
MOVZ.*fmt*	GPR 为 0 时复制 FPR	MIPS32
MTC1	从 GPR 复制低字到 FPR	MIPS32
MTHC1	从 GPR 复制高字到 FPR	MIPS32 R2

GS464 与 MIPS64 Release 2 版本兼容，从功能上实现了 MIPS64 体系结构规定的所有 FPU 指令，但是有些指令在实现上有细微的并不影响兼容性但是比较重要的差别，以下两点值得编程人员注意

1. 乘加、乘减指令。在执行 MADD.fmt，MSUB.fmt，NMADD.fmt，NMSUB.fmt 这四组指令时，GS464 的运算结果与 MIPS64 处理器略有不同，这是因为 GS464 在做乘加运算时只在最后结果处做精度舍入，而 MIPS64 处理器在进行乘运算后就进行了一次舍入，加运算后又做了一次舍入，导致了最终结果最低位相差 1。

2. 单精度运算指令。在 Status 控制寄存器的 FR 位为 0 时，abs.s, add.s, ceil.w.d, ceil.w.s, div.s, floor.w.d, floor.w.s, mul.s, neg.s, round.w.d, round.w.s, sqrt.s, sub.s, trunc.w.d, trunc.w.s, mov.s, cvt.d.s, cvt.d.w, cvt.s.d, cvt.s.w, cvt.w.d, cvt.w.s, movf.s, movn.s, movt.s, movz.s 等 26 条指令不能使用奇数号寄存器，而 MIPS64 体系结构的处理器就可以，在这点上龙芯沿用了 MIPS R4000 与 MIPSR10000 的做法，与 MIPS64 的规定略有不同。（早期的 MIPS 处理器中 FR 位表示浮点寄存器是 16 个还是 32 个，MIPS64 中 FR 位表示浮点寄存器是 32 位还是 64 位)。

7.3 浮点部件格式

7.3.1 浮点格式

GS464 的浮点部件不仅可以处理符合 IEEE 标准的单精度（32 位）及双精度（64 位）浮点数，同时也提供了对“单精度对”（Paired single, 简称 PS）浮点格式的支持。图 7.2 显示了这三种浮点数的详细格式。

(FIXME)

图 7.2: 浮点格式

如图所示，浮点数的格式由以下三个域组成：

- 符号域， S；
- 带偏移的指数域， $E = E_0 + Bias$，其中 E_0 是不带偏移的指数
- 小数域， $f = .b_1 b_2 .... b_{p-1}$

其中指数 E_0 的可能取值为在 $[E_{min}, E_{max}]$ 之间的任何整数。另外还有两个保留的指数值用于编码特殊数值：

- $E_{min} - 1$：用来编码 ± 0 和非规范化数（denormalized）；
- $E_{max} + 1$：用来编码 $\pm\infty$ 和 NaN （非数，Not a Number）。

对于单精度或者双精度格式来说，每一个非 0 数都只有唯一一种编码与之对应。一个编码所对应的数值 V 可以由表 7.8 中的公式所决定。

E	$f \neq 0$	$f = 0$
$E_{max} + 1$	NaN	$(-1)^s \infty$
$[E_{min}, E_{max}]$	$(-1)^s 2^E 1.f$	
$E_{min} - 1$	$(-1)^s 2^{E_{min}} 0.f$	$(-1)^s 0$

表 7.8: 单双精度浮点数：数值公式

对于所有的浮点格式，如果 V 是一个 NaN，那么小数域 f 的最高位决定了这个数是 Signaling NaN （SNAN）还是 Quiet NaN（QNAN）：如果 f 的最高位被设置，那么 V 是 QNAN，否则为 SNAN。表 7.9 给出了定义浮点格式的相关参数值，表 7.10 列出了可表达的浮点的最大值和最小值。

7.4 FPU 指令流水线概述

FPU 提供一个和 CPU 指令流水线并行的指令流水线。它和 CPU 共享基本的 9 级流水线体系结构，但根据浮点操作的不同，执行流水级又细分为 2~6 个流水级。每个 FPU 指令被两个浮点功能单元中的一个执行：FALU1 或者 FALU2。FALU1 可以执行所有的浮点运算操作及媒体操作。FALU2 仅执行浮点加减、乘法、乘加操作以及所有媒体操作。

参数	格式	
	单精度	双精度
E_{max}	+127	+1203
E_{min}	-126	-1022
指数偏移量（$Bias$）	+127	+1023
指数域，E，宽度	8	11
小数域，f，宽度	24	53
总宽度	32	64

表 7.9: 浮点格式参数值表

类型	最小数	最小正规数	最大数
单精度	$1.40129846e^{-45}$	$1.17549435e^{-38}$	$3.40282347e^{38}$
双精度	$4.9406564584124654e^{-324}$	$2.2250738585072014e^{-308}$	$1.7976931348623157e^{308}$

表 7.10: 浮点范围值表

每个 FALU 单元每个周期能够分别接收 1 条指令，并能向浮点寄存器文件分别送出一个结果。在每个 FALU 单元中，浮点加减、浮点乘法、浮点乘加运算需要 6 个执行周期；定点与浮点间的格式转换运算需要 4 个执行周期；浮点除法根据操作数的不同需要 4~16 个执行周期；浮点开平方根根据操作数的不同需要 4~31 个执行周期，其它浮点运算需要 2 个执行周期。在每个 FALU 单元中，如果两条有着不同执行周期的指令在同一拍输出结果，执行周期较短的指令优先向总线输出结果。其中除浮点除法和浮点开根之外的浮点操作和所有媒体操作都是全流水的。如果同时有两个浮点除法指令或者两个浮点开平方根指令在 FALU1 中，那么 FALU1 单元将向前一流水级发出一个停顿信号，并且 FALU1 单元在除法或开平方根指令写回前不能接收新的指令。

7.5 浮点例外处理

该节描述了浮点计算的例外。浮点例外发生在当 FPU 不能以常规的方式处理操作数或者浮点计算的结果时，FPU 产生相应的例外来启动相应的软件陷阱或者是设置状态标志位。

FPU 的控制和状态寄存器对于每一种例外都包含一个使能位，使能位决定一个例外是否能够导致 FPU 启动一个例外陷阱或者设置一个状态标志。如果一个陷阱启动，FPU 保持操作开始的状态，启动软件例外处理路径；如果没有陷阱启动，一个适当的值写到 FPU 目标寄存器中，计算继续进行。

FPU 支持五个 IEEE754 定义的标准例外：

- 不精确例外（Inexact，I）；
- 下溢例外（Underflow，U）；
- 上溢例外（Overflow，O）；
- 除零例外（Division by Zero，Z）；
- 非法操作例外（Invalid Operation，V）；

以及

- 未实现操作例外（Unimplemented Operation，E）。

未实现操作例外用于 FPU 不能执行标准的 MIPS 浮点结构，包括 FPU 不能决定正确的例外行为的情况。该例外指示了软件例外处理的执行。未实现操作例外没有使能信号和标志位，当这个例外发生时，一个相应的未实现例外陷阱发生。

IEEE754 定义的 5 个例外（V，Z，O，U，I）都对应着一个用户控制的例外陷阱，当 5 个使能位的某一位被设置时，相应的例外陷阱被允许发生。当例外发生时，相应的导致（Cause）位被设置。如果相应的使能（Enable）位没有设置，例外标志（Flag）位被设置；如果使能位被设置，那么标志位将不被设置，同时 FPU 产生一个例外给 CPU – 随后的例外处理允许该例外陷阱发生。

当没有例外陷阱信号时，浮点处理器采取缺省方式进行处理，提供一个浮点计算例外结果的替代值。不同的例外类型使用不同的缺省值。表 7.11 列出了 FPU 对于每个 IEEE 例外的默认处理。

域	描述	舍入模式	默认操作
I	非精确例外	任何	提供舍入后的结果
U	下溢例外	RN	根据中间结果的符号把结果置 0
		RZ	根据中间结果的符号把结果置 0
		RP	把正下溢修正为最小正数，把负下溢修正为 -0
		RM	把负下溢修正为最小负数, 把正下溢修正为 $+0$
O	上溢例外	RN	根据中间结果的符号把结果置为 ∞
		RZ	根据中间结果的符号把结果置为最大数
		RP	把负下溢修正为最大负数, 把正下溢修正为 $+\infty$
		RM	把正下溢修正为最大整数, 把负下溢修正为 $-\infty$
Z	除零例外	任何	提供一个相应的带符号的无穷大数
V	非法操作例外	任何	提供一个 Quiet Not a Number(QNaN)

表 7.11: 例外的默认处理

下面对导致 FPU 产生每种例外的条件进行了描述，并且详细说明了 FPU 对每个例外导致条件的反应。

7.5.1 不精确例外 (I)

FPU 在发生如下的情况时产生不精确例外：舍入结果非精确舍入结果上溢舍入结果下溢，并且下溢和不精确的使能位都没有被设置，而且 FS 位被设置。

结果（陷阱被使能）： 如果一个非精确例外陷阱被使能，结果寄存器不被修改，并且源寄存器被保留。因为这种执行模式会影响性能，所以不精确例外陷阱只有在必要的时候才被使能。

结果（陷阱未使能）： 舍入或者上溢结果被发送到目标寄存器。

7.5.2 下溢例外 (U)

两个相关的事件导致了下溢例外：一个很小的在 ±2Emin 之间的非零结果，由于该结果非常小，因此会导致其后发生下溢例外。用非规范化数据（Denormalized Number）来近似表示这两个

小数据所产生的严重的数据失真。IEEE754 允许用多种不同的方法检测这些事件，但对于所有的操作要求用相同的方法来检测。小数据可以用下面的方法的一种来检测：舍入后（如果一个非零的数据，在指数范围没有界限的情况下来计算，应该严格的位于 ±2Emin 之间）舍入前（如果一个非零的数据，在指数和精度范围没有界限的情况下来计算，应该严格的位于 ±2Emin 之间）MIPS 的结构要求微小数据在舍入后检测。精度失真可以用如下方法的一种来检测：非规范化数据的失真（当产生的结果与指数没有界限时计算的结果不同）非精确数据（当产生的结果与指数和精度范围没有界限的情况下计算的结果不同）MIPS 结构要求精度失真被检测为产生非精确结果。

结果（陷阱被使能）： 如果下溢或者不精确例外被使能，或者 FS 位没有设置，产生未实现操作例外，结果寄存器不被修改。

结果（陷阱未使能）： 如果下溢或者不精确例外不被使能，而且 FS 位被设置，最后的结果由舍入模式和立即结果的符号位来决定。

7.5.3 上溢例外 (O)

当舍入后的浮点结果的幅度用没有界限的指数来表示时，大于最大的目标模式所表示有限数据，上溢例外发出通知信号。（这个例外同时设置不精确例外和标志位)

结果（陷阱被使能）： 结果寄存器不被修改，源寄存器保留。

结果（陷阱未使能）： 如果没有陷阱发生，最后的结果由舍入模式和中间结果的符号来决定。

7.5.4 除零例外 (Z)

除法运算中当除数是 0 被除数是一个有限的非零的数据时，除零例外发出信号通知。利用软件可以对其他操作产生有符号的无穷值时模拟除零例外，ln(0)，如：sin(/2)，cos(0)，或者 0-1。

结果（陷阱被使能）： 结果寄存器不被修改，源寄存器保留。

结果（陷阱未使能）： 如果没有陷阱发生，结果是有符号的无穷值。

7.5.5 非法操作例外 (V)

当一个可执行的操作的两个操作数或其中的一个操作数是非法时，非法操作例外发出信号通知。如果例外没有陷入，MIPS 定义这个结果是一个 Quiet Not a Number （QNaN）。非法操作包括：加法或者减法：无穷相减。例如：(+ ∞)+(- ∞) 或者 (-∞)-(- ∞) 乘法：0× ∞，对于所有的正数和负数除法：0/0，∞/∞，对于所有的正数和负数当不处理 Unordered 的比较操作的操作数是 Unordered 对一个指示信号 NaN 进行浮点比较或者转换任何对 SNaN（Signaling NaN）的数学操作。当其中一个操作数为 SNaN 或者两个都为 SNaN 时会导致这个例外（MOV 操作不被认为是数学操作，但 ABS 和 NEG 被认为是数学操作）开方：

X ，当 X 小于 0 时

软件可以模拟其他给定源操作数的非法操作的例外。例如在 IEEE754 中利用软件来实现的特定函数：X REM Y，这里当 Y 是 0 或者 X 是无穷的时候；或者当浮点数转化为十进制时发生上溢，是无穷或者是 NaN；或者先验函数例如：ln(5) 或者 cos-1(3)。

结果（陷阱被使能）： 源操作数的值不被发送。

结果（陷阱未使能）： 如果没有其他例外发生，QNaN 被发送到目标寄存器中。

7.5.6 未实现操作例外 (E)

当执行任何一条为以后定义所保留的操作码或者操作格式指令时，FPU 控制/状态寄存器中的未实现操作导致位被设置并产生陷阱。源操作数和目的寄存器保持不变，同时指令在软件中仿真。IEEE754 中的任何一个例外都能够从仿真操作中产生，这些例外反过来可以被仿真。另外，当硬件不能正确执行一些罕见的操作或者结果条件时，也会产生未实现指令例外。这些包括:

- 非规范化操作数（Denormalized Operand），比较指令除外
- Quite Not a Number 操作数（QNaN），比较指令除外
- 非规范化数据或者下溢，而且当下溢或者不精确使能信号被设置同时 FS 位没有被设置

注意: 非规范化和 NaN 操作只在转换或者计算指令中进入陷阱，在 MOV 指令中不进入陷阱。陷阱被使能的情况：原操作数据不被发送。该陷阱不能被屏蔽。

第八章　性能优化

本章提供了龙芯 GS464 体系结构中一些与软件性能优化相关的信息，包括指令延迟和循环间隔、扩展指令、指令流和存储访问处理等，供编译器和其它软件开发者参考。

8.1　用户指令延迟和循环间隔

操作	执行单元	延迟	循环间隔
整型操作			
ADD/SUB/LOGICAL/SHIFT/LUI/CMP	ALU1/2	2	1
TRAP/BRANCH	ALU1	2	1
MF/MT HI/LO	ALU1/2	2	1
(D)MULT(U)	ALU2	5	21
(D)MULT(U)G	ALU2	5	1
(D)DIV(U)	ALU2	5-38	10-763
(D)DIV(U)G	ALU2	5-38	4-37
(D)MOD(U)G	ALU2	5-38	4-37
取	MEM	5	1
存	MEM	-	1
浮点操作			
(D)MTC1/(D)MFC1	MEM	5	1
ABS/NEG/C.COND/BCLT/BC1F/MOVE/CVT*	FALU1	3	1
ROUND/TRUNC/CEIL/FLOOR/CVT*	FALU1	5	1
ADD/SUB/MUL/MADD/MSUB/NMADD/NMSUB	FALU1/2	7	1
DIV.S	FALU2	5-11	4-10
DIV.D	FALU2	5-18	4-17
SQRT.S	FALU2	5-17	4-16
SQRT.D	FALU2	5-32	4-31
LWC1/LDC1	MEM	5	1
SWC1/SDC1	MEM	-	1

表 8.1:

表 8.1 给出了在 ALU1/2, MEM, FALU1/2 功能单元中执行的所有用户指令的延迟和循环间隔（不包括内核指令和控制指令）。这里的指令延迟是指从该指令发射到其结果能被下一条指令使用所需要的拍数（一个处理器周期为一拍）。例如，大部分的 ALU 指令延迟为 2，这表示 ALU 指

令的结果要隔一拍后才能被后续指令使用。因此，形如 $i = i + 1$ 的相关循环（下一个循环依赖上一个循环的结果）不能每拍出一个结果。而一个指令的循环间隔则是指功能部件接受这种指令的频度，1 表示每拍都能接受一个以上的同类指令， n 表示功能部件接受一个该指令后，需要等 $n - 1$ 拍后才能再接受同类指令。全流水功能部件的指令循环间隔为 1。

对于表 8.1 中的数据

- 内存存取操作的循环间隔不包括 LL/SC 指令。这是因为 LL/SC 是等待发射操作，只有当它们位于 Reorder 队列队首，而且 CP0 队列为空时，才可以被发射。
- 对于 HI/LO 寄存器，没有特别的使用限制：它们可以当作通用寄存器一样被使用。
- 表中数据也不包含 CTC1/CFC1：它们和许多其它的控制指令一样被序列化。
- 表中数据也不包含多媒体指令。这是因为它们是通过扩展普通浮点指令的格式而完成的，它们的功能单元和延时与被扩展的指令相同。

8.2 指令扩充

龙芯 GS464 完成了以下几种指令扩充：

- 只写一个结果到通用寄存器的定点乘除。包括 12 条指令：

  ```
  (D)MULT.G,  (D)MOD.G,  (D)DIV.G,
  (D)MULTU.G, (D)MODU.G, (D)DIVU.G
  ```

  在标准的 MIPS 指令集中，乘法和除法在一个操作中需要写两个特殊的结果寄存器 (HI/LO)，它们在 RISC 流水线中很难实现。为了使用这些结果，将不得不使用额外的指令把它从 HI/LO 中取出送入通用寄存器中。更麻烦的是，由于流水线的问题，很多 MIPS 处理器对这些指令的使用还有些限制。这些新指令执行速度更快，同时也更容易使用。

- 多媒体指令的扩充。它们是定点操作但使用浮点的数据通路。在执行定点程序的过程中，浮点数据通路常常处于空闲状态，这些指令使得我们有机会利用它们，进一步增加指令并行的程度。

8.3 指令流

龙芯 GS464 是一个多发射高度并行的处理器，对本质上是串行的指令流的处理可能会对程序性能产生明显的影响，本节讨论关于指令对齐、转移指令、指令调度等问题。

8.3.1 指令对齐

在一个周期内，龙芯 GS464 可以从一个 Cache 行中取出四条指令，但这四条指令不能跨越 Cache 行的边界。我们应该对那些经常性被执行的基本块进行合适的对齐，以避免跨越 Cache 行的边界。此外，如果在一次所取的四条指令中存在转移指令，也会影响取指的效率。如果第一条是转移分支指令，而且转移预测是成功，那么最后两条指令将被抛弃。如果最后一条是转移指令，即使转移成功，处理器也不得不再取下一个 Cache 行以得到它的延迟槽中的指令。龙芯 GS464 一个周期只能给一条转移指令译码，如果在一束指令中存在两条转移指令的话，它将需要两个周期来完成译码，也就是说取指引擎将被阻塞一个周期。

8.3.2 转移指令的处理

在龙芯 GS464 的处理器中，指令流地址的一个意想不到的变化会浪费大约 10 条指令的时间。“意想不到”可以是由转移成功的指令导致，也可能会由转移预测错误导致。对于目前的龙芯 GS464 而言，即使是一个正确预测且预测转移成功的转移指令也比顺序代码慢，它会浪费一个周期，因为对于普通条件转移指令，转移目标缓存器（BTB）不能给出下一个正确的程序计数器 PC 值。

编译器可以通过以下的方法来减少转移指令引起的开销:

- 龙芯 2 号的转移指令预测方法和其它的高性能处理器都不同，且不同的版本都有一些细微的差别。基于执行剖析（Profile)，编译器可以根据实际的转移频率对代码位置进行重新安排，从而得到较好的预测结果。
- 尽可能使基本块变大。一种比较好的优化结果就是使得在两条转移成功的转移指令之间平均有 20 条指令。为了使它们之间至少含有 20 条指令，这就需要循环展开，还要把不到 20 条指令的子程序直接内联展开。龙芯 GS464 实现了条件性移动指令，它可以用来减少分支指令数量。通过执行剖析来重新组织代码也有助于这个优化。

1.3 节给出了取指译码单元的一个概要描述。正如我们所见，不同的转移指令使用不同的方式进行预测:

- 静态预测。针对 likely 类转移指令和直接跳转指令。
- G-Share 预测器。一个 9 位的全局历史寄存器 GHR，和一个有 2K 项的模式历史表 PHT。用于条件转移指令。
- BTB（转移目标缓存)。有 16 项的全相联的缓存。被用于预测寄存器跳转指令的目标地址。
- RAS（返回地址栈)。4 项，被用于预测函数返回的目标地址。

以下有几点关于软件需要注意的地方:

在龙芯 GS464 处理器上需要特别小心地使用 Likely 类转移指令。尽管 Likely 类转移指令也许对顺序标量处理器的简单的静态调度很有效，但是它对现代高性能处理器并不是同样有效。因为现代高性能处理器的转移预测硬件是比较复杂，它们通常有 90% 以上的正确预测率。(比如说，龙芯 GS464 能够正确预测 85%-100%，平均 95% 的条件转移的转移方向）在这种情况下，编译器不应该使用预测率不太高的 Likely 类转移指令。事实上，我们发现带有 `-mno-branch-likely` 选项的 GCC（3.3 版）通常会工作得更好。

取指译码单元被划分成 3 个流水段，其中转移的目标地址在第三阶段被计算。转移成功的转移指令将会导致有两个周期的停顿，也就是说，如果在周期 0 取出一条转移指令，在周期 1 取出地址为 PC+16 的指令，周期 2 取出地址为 PC+32 的指令，在周期 3 时，才会取到转移指令的目标地址。所以减少转移成功的转移指令数将会比较有一些帮助。

龙芯 GS464 中的 BTB 仅被用于寄存器跳转指令 (包括 JALR 及除 JR31 外的 JR 指令)。通过一个 4 项的 RAS 来预测 JR31 指令的目标地址。函数返回的预测有效性取决于那些使用 JR31 指令作为函数返回指令的软件。

8.3.3 指令流密度的提高

编译器应该尽量利用执行剖析，以确保调入指令 Cache 的那些字节均被执行。这就要求跳转指令的目标地址需对齐，并且把那些很少被执行的代码移出 Cache 行。

8.3.4 指令调度

龙芯 GS464 内部有比较大的指令窗口会进行动态的指令调度，但是由于处理器内部各种资源有限无法做到最优的调度，编译器可以在一定程度上协助处理器进行更好的调度。现代的编译器（如 GCC）有指令调度的支持，把龙芯 GS464 内部的部件资源情况和指令的延迟情况告诉编译器，它就能够进行较好的调度。

8.3.5 存储器访问

Load/Store 指令的执行对整个系统性能有很大的影响。如果一级数据 Cache 中包括所需的内容，那么这些指令可以很快被执行。如果数据只在二级 Cache 则稍微慢些（需要增加额外的 11 拍），如果只在主存中则会有很大的延迟。不过，乱序执行和非阻塞 Cache 可以减少由这些延时带来的性能损失。

龙芯 GS464 的二级 Cache 存放指令和数据，容量是 512KB，组织为四路组相联。它工作在处理器主频同样的频率，采用非阻塞结构，即每拍可以访问一次。龙芯 GS464 内置 DDR2 内存控制器，最大限度地减少了内存访问的延迟。与龙芯 2E 不同，龙芯 GS464 的内存工作频率是独立设置，与处理器主频没有关联，因此更有利于发挥内存的性能。有关内存控制器更多的信息可以参考第十章。

龙芯 GS464 目前的这个版本没有直接提供预取指令，但是可以通过 Load-to-Zero-Register 来获得预取的功能，即向 0 号寄存器取一个数。0 号寄存器值永远为 0，因此这样的指令不会影响程序员可见的状态，但可以迫使一些数据进入 Cache。为了降低开销，这种指令执行的时候不会发生访问异常，即使地址非法也会被悄悄地忽略。

编译器应该尽量减少不必要的存储访问。目前的龙芯 GS464 处理器的存储指令延迟较大（即使是 Cache 命中，也需要 5 个周期），同时指令窗口也没有大到可以容忍几十周期的访问延迟。

软件还要特别注意数据对齐的问题。集合体（数组，一些纪录，子程序堆栈帧）应该被分配在对齐的 Cache 行边界上，这样就可以利用 Cache 行对齐数据通路，还可以降低 Cache 行被填满的数目。在那些强迫不对齐（例如 GCC 的 Packed 属性）的集合体（纪录，普通块）中的项目中，应该产生一个编译时间的警告信息。在龙芯 GS464 中正常的 Load/Store 指令有对齐要求，不满足要求的存储访问或者采用非对齐访存指令对来访问或者通过内核模拟来实现。例如，从非四字节对齐的地址取一个字（四字节）会触发例外，由操作系统来处理；通常操作系统需要几千个处理器周期才能完成这个任务。因此用户需要知道这些警告信息代表代码的性能可能会很低。编译参数的代码都默认这些参数是对齐的。那些经常被使用的标量应该驻留在寄存器中。

8.4 其他提示

- 使用所有的浮点寄存器。尽管 O32 ABI 只开放了 16 个给用户使用，但是龙芯 GS464 提供了 32 个 64 位的浮点寄存器。使用 N32 或 N64 ABI 有助于发挥处理器的性能。

- 使用性能计数器。龙芯 GS464 的性能计数器可以用来监控程序的实时性能参数。编译器和软件开发者可以通过分析这个结果来改进他们的代码。

第九章　MIPS 兼容性

龙芯 GS464 处理器的设计目标是要兼容 MIPS64 R2 架构，这个目标已经基本实现。在设计过程，出于对性能、将来扩展的考虑以及具体实现的原因，龙芯 GS464 核与 MIPS64 R2 相比在细节上有一些不太重要的差异上。同时，GS464 在 MIPS64 R2 的基础上实现了若干扩展。以下，我们将从指令集构架和特权资源构架两个方面对这些差异进行阐述。

9.1　指令集架构

在指令集架构（Instruction Set Architecture，简称 ISA）上，龙芯 GS464 处理器的实现了 MIPS64 R2 的所有指令，不过在一些指令的实现上做了重定义，具体细节见9.1.1 节。同时，龙芯 GS464 处理器在 MIPS64 R2 的基础上实现了若干扩展。这些增强指令包括如下五个方面：

- 扩展定点乘除法指令，扩大了 MIPS64 的定点乘除法指令，以提高定点乘法运算使用广泛的应用性能。这些指令主要执行 64 位整点乘除法操作，并产生 64 位（而不是 128 位）结果。附录 A 对龙芯 GS464 的扩展整数指令进行了详细介绍。
- 扩展浮点指令，这些指令采用了三个操作数模式，而不是 MIPS64 中的四操作数形式。附录 B 龙芯 GS464 扩展浮点指令进行了详细介绍。
- 扩展多媒体指令，即单指令多数据（SIMD）指令，对高性能媒体和通信应用提供了强大支持。这些指令类似于 x86 平台的 SSE 的指令。附录 C 对龙芯 GS464 的多媒体指令进行了详细介绍。
- 扩展访存指令 – to be added.
- x86 虚拟指令 – to be added.

9.1.1　CPU 特殊实现指令列表

龙芯 GS464 处理器核对所有 MIPS64 R2 指令都作了支持。但对一些与实现相关的指令作了重定义，见表

表 9.1: 龙芯 CPU 特殊实现指令

指令 OpCode	描述	龙芯具体实现
PREF	预取指令	空操作，预取可通过 Load 到 0 号寄存器实现
PREFX	预取指令	空操作，预取可通过 Load 到 0 号寄存器实现

未完待续

表 9.1: 龙芯 CPU 特殊实现指令（续）

指令 OpCode	描述	龙芯具体实现
SSNOP	单发射空操作	空操作，GS464 实现了所有相关指令的硬件互锁
EHB	隔离执行相关	空操作，GS464 实现了所有相关指令的硬件互锁
WAIT	进入等待状态	空操作，GS464 实现了通过动态调频，让 CPU 等待或休眠
RDPGPR	读影子寄存器	GS464 没有影子寄存器，读当前寄存器
WRPGPR	写影子寄存器	GS464 没有影子寄存器，写当前寄存器

9.1.2 浮点转换指令

龙芯 GS464 浮点运算中进行数据格式转换时，比如使用如下指令将浮点数转换为字整点数或双字整点数，

```
cvt.w.fmt, round.w.fmt, floor.w.fmt, ceil.w.fmt, trunk.w.fmt
cvt.l.fmt, round.l.fmt, floor.l.fmt, ceil.l.fmt, trunk.l.fmt
```

如果输入值为负非数，负越界，负无穷时，而 FCSR 寄存器的无效例外没有使能，将不会发出无效操作例外，返回值为负最大：0x8000_0000 或 0x8000_0000_0000_0000 取决于操作的转换格式。而在 MIPS64 浮点运算中，遇到相同情况时，也不会发出无效操作例外，不过返回值规定为正最大：0x7fffffff 或 0x7fffffffffffffff。即，MIPS64 的规定不区分输入值的正负。

9.2 特权资源构架

龙芯 GS464 处理器实现了 MIPS64 特权资源架构的绝大部分，而在一些地方则采用了 R10000 的特权资源架构，同时也引入了一些特殊的实现特征 — 这些特征和 MIPS64 的特权资源架构是不兼容的。为了实现的灵活性，MIPS 在实践中也允许一个兼容设计只实现部分的特权资源架构。作为 MIPS64 定义的特权资源架构的重要部分，这些差异主要体现在 CP0 上。系统程序员在移植或编写软件时应充分考虑到这些不兼容的特征。以下各节将更详细地描述这些主要的特征差异。

9.2.1 Diagnostic 寄存器

Diagnotic 寄存器是一个 64 位的辅助寄存器。该寄存器具有刷新 ITLB 、BTB （分支目标缓冲器）和启用 RAS（返回地址堆栈）的功能。当在 Diagistic 寄存器的 ITLB 位写入 1 时，整个 ITLB 将被刷新。

Remark: BTB and RAS: any further comments?

9.2.2 ITLB 刷新

龙芯 GS464 包含了独立的 ITLB（即指令 TLB），以减少对 JTLB 的竞争及降低功耗。当翻译指令地址发生 ITLB 查找脱靶时，将从 JTLB 中查找命中项，并随机替换一个 ITLB 项。这里的 ITLB 的查找和替换操作对用户是完全透明的。然而，GS464 处理核没有提供自动刷新 ITLB 的功能：当 JTLB 表项映射的地址被 TLBWI 或 TLBWR 指示改变时，ITLB 不能自动更新以保

持与 JTLB 的一致。因此，系统程序员必须使用指令刷新 ITLB。基于同样的原因，当任何 JTLB 项的位域被改变时，比如失效一个页面或页面的掩码大小发生变化时，也必须用软件刷新 ITLB。

刷新 ITLB 是通过设置 CP0 的 Diagistic 寄存器的 ITLB 位实现的。ITLB 项不能单独地被刷新：这个操作将丢弃 ITLB 中的所有项。以下给出了一个刷新 ITLB 的例子。

```
/* 参数含义：
   t4 - 32 bit Virtual address
   t3 - ASID value
   t0 - 32 bit physical address for EntryLo1 (待创建的 JTLB 项)
   t1 - 32 bit physical address for EntryLo0 (待创建的 JTLB 项)
   attr0 - TLB attribute for EntryLo0
   attr1 - TLB attribute for EntryLo1
   t5 - Index value of the TLB entry
   t6 - temp register             */

srl      t6, t4, 13;          /*  Clean up lower order bits   */
sll      t6, t6, 13;          /*  Pad zeros                   */
or       t6, t6, t3;          /*  Include the ASID value      */
mtc0     t6, C0_EntryHi;      /*  Write to entry Hi register  */
srl      t6, t1, 6;           /*  align PFN for entry Lo reg  */
sll      t6, t6, 6;
ori      t6, t6, attr0;       /* Include the attribute field  */
mtc0     t6, C0_EntryLo0;     /* Write to entry lo0 reg       */
srl      t6, t0, 6;           /* align PFN for entry Lo reg   */
sll      t6, t6, 6;
ori      t6, t6, attr1;       /*  Include the attribute field */
mtc0     t6, C0_EntryLo1;     /*  Write to entry lo1 reg      */
mtc0     t5, C0_Index;        /*  Write to Index register     */
tlbwi;                        /*  Do a TLB write              */
li       k1, (0x1 << 2);      /*  Set ITLB flushing bit       */
mtc0     k1, C0_Diag          /*  Write to Diagnostic reg     */
```

9.2.3 异常返回（Status[ERL] = 1）

ERET 指令用于从异常，中断或错误陷阱中返回。当复位，软复位，NMI 或缓存错误异常发生时，处理器将设置 ERL 位。对于 MIPS64 兼容处理器，在 Status[ERL]=1 的情况下，ERET 指令将清零 ERL 位并从 ErrorEPC 寄存器的值返回。龙芯 GS464 处理器的处理略有不同：ERL 位被设置时，ERET 指令将会清除该位，并依照 EPC （而不是 ErrorEPC）寄存器给出的地址返回。

Remark: is it still like this now?

9.2.4 64 位地址空间

MIPS64 兼容处理器一般同时支持 32 位和 64 位地址空间：不同的操作模式可以通过 CP0 状态寄存器的 UX、SX 和 KX 位设置。而龙芯 GS464 处理器的 64 位地址空间总是启用的。也就

是说，状态寄存器的 KX、SX 或 UX 位的值始终为 1，软件不能将它们写为 0。所以无论处理器运行在何种模式下，用户空间（XUSEG、XSUSEG 和 XKUSEG 段），管理空间（XSSEG 和 XKSSEG 段）及内核空间（KXSEG 和 KXPHYS 段）始终有效。内存管理一章的 4.2 节对这些虚拟地址段给出了详细的描述。

由于龙芯 GS464 处理器只能工作在 64 位地址空间下，所以 TLB 重填例外只能使用 XTLB 例外向量。XTLB 重填例外在龙芯 GS464 处理器上有和 TLB 重填向量相同的位置，即偏移量为 0x00。这是龙芯 GS464 处理器与其他 MIPS64 兼容处理器的重要差别。

项目	GS464	MIPS64
TLB 例外	龙芯 GS464 不支持 KX,UX,SX 位，XTLB 例外入口与 TLB 例外入口相同	MIPS64 中用 KX,UX,SX 位来区分 64 位地址和 32 位地址，且 TLB 例外和 XTLB 例外分别使用不同的例外入口。

附录 A　龙芯新增整型指令

在标准的 MIPS 指令集中，整点乘法和除法在一个操作中将结果写入两个特殊的寄存器（HI/LO）中，而它们在 RISC 流水线中很难实现。为了使用这些结果，往往必须通过额外的指令把它从 HI/LO 中取出送到通用寄存器中。更麻烦的是，由于流水线问题，很多 MIPS 处理器对这些指令的使用还有额外的限制。相比之下，这些新增的指令执行速度更快，没有流水线问题，同时也更容易使用。

A.1　新增指令详解

A.1.1　MULT.G （龙芯字乘）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	MULT.G 0 1 0 0 0 0
6	5	5	5	5	6

格式： `MULT.G rd, rs, rt`

功能： 32 位有符号整数乘。

描述： 通用寄存器 rs 中 32 位值乘以通用寄存器 rt 中 32 位值，这两个操作数都是有符号数，产生一个 64 位结果。结果的低 32 位保存在目的寄存器 rd 中。

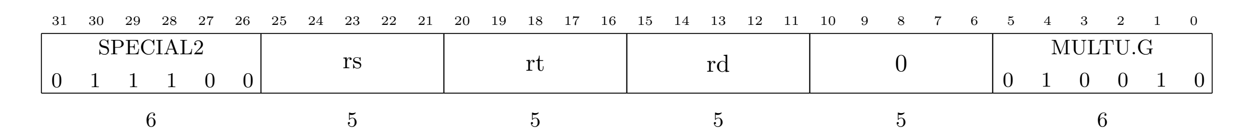
任何情况下都不会产生算术异常。

操作：

```
prod ← rs[31..0] * rt[31..0]; /* 有符号乘 */
rd   ← sign_extend(prod[31..0]);
```

例外： 无。

A.1.2　MULTU.G （龙芯无符号字乘）



31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	MULTU.G 0 1 0 0 1 0
6	5	5	5	5	6

格式： `MULTU.G rd, rs, rt`

功能： 32 位无符号整数乘。

描述： 通用寄存器 rs 中 32 位值乘以通用寄存器 rt 中 32 位值，这两个操作数都是无符号数，产生一个 64 位结果。结果的低 32 位保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：

```
prod ← rs[31..0] * rt[31..0]; /* 无符号乘 */
rd   ← sign_extend(prod[31..0]);
```

例外： 无。

Remark: 这里有符号扩展？

A.1.3 DMULT.G （龙芯双字乘）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	DMULT.G 0 1 0 0 0 1
6	5	5	5	5	6

格式： `DMULT.G rd, rs, rt`

功能： 64 位有符号整数乘。

描述： 通用寄存器 rs 中 64 位值乘以通用寄存器 rt 中 64 位值，这两个操作数都是有符号数，产生一个 128 位结果。结果的低 64 位保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：

```
prod ← rs * rt; /* 有符号乘 */
rd   ← prod[63..0];
```

例外： 无。

A.1.4 DMULTU.G （龙芯无符号双字乘）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	DMULT.G 0 1 0 0 0 1
6	5	5	5	5	6

格式： `DMULTU.G rd, rs, rt`

功能： 64 位无符号整数乘。

描述： 通用寄存器 rs 中 64 位值乘以通用寄存器 rt 中 64 位值，这两个操作数都是无符号数，产生一个 128 位结果。结果的低 64 位保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：

```
prod ← rs * rt; /* 无符号乘 */
rd   ← prod[63..0];
```

例外： 无。

A.1.5 DIV.G （龙芯字除）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	DIV.G 0 1 0 1 0 0
6	5	5	5	5	6

格式： `DIV.G rd, rs, rt`

功能： 32 位有符号整数除。

描述： 通用寄存器 rs 中 32 位值除以通用寄存器 rt 中 32 位值，这两个操作数都是有符号数。32 位商保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：

```
q  ← rs[31..0] div rt[31..0]; /* 有符号除 */
rd ← sign_extend(q[31..0]);
```

例外： 无。

A.1.6 DIVU.G （龙芯无符号除）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	DIVU.G 0 1 0 1 0 0
6	5	5	5	5	6

格式： `DIVU.G rd, rs, rt`

功能： 32 位无符号整数除。

描述： 通用寄存器 rs 中 32 位值除以通用寄存器 rt 中 32 位值，这两个操作数都是无符号数。32 位商保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：

```
q  ← rs[31..0] div rt[31..0]; /* 无符号除 */
rd ← sign_extend(q[31..0]);
```

例外： 无。

A.1.7 DDIV.G （龙芯双字除）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	DDIV.G 0 1 0 1 0 1
6	5	5	5	5	6

格式： `DDIV.G rd,rs, rt`

功能： 64 位有符号整数除。

描述： rd ← rs / rt

通用寄存器 rs 中 64 位值除以通用寄存器 rt 中 64 位值，这两个操作数都是有符号数。64 位商保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：

```
rd ← rs div rt; /* 有符号除 */
```

例外： 无。

A.1.8 DDIVU.G （龙芯无符号双字除）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	DDIVU.G 0 1 0 1 1 1
6	5	5	5	5	6

格式： `DDIVU.G rd, rs, rt`

功能： 64 位无符号整数除。

描述： 通用寄存器 rs 中 64 位值除以通用寄存器 rt 中 64 位值，这两个操作数都是无符号数。64 位商保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：

```
rd ← rs div rt; /* 无符号除 */
```

例外： 无。

A.1.9 MOD.G （龙芯字求模）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	MOD.G 0 1 1 1 0 0
6	5	5	5	5	6

格式： `MOD.G rd, rs, rt`

功能： 32 位有符号整数求模。

描述： 通用寄存器 rs 中 32 位值除以通用寄存器 rt 中 32 位值，这两个操作数都是有符号数。32 位剩余数保存在目的寄存器 rd 中。

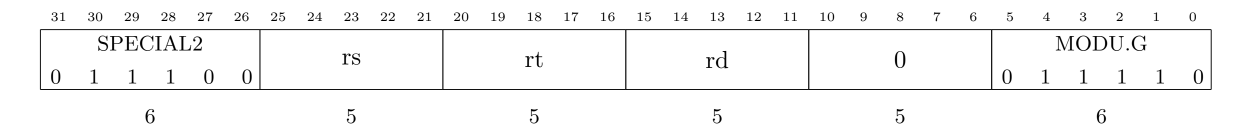
任何情况下都不会产生算术异常。

操作：

```
q  ← rs[31..0] mod rt[31..0]; /* 有符号求模 */
rd ← sign_extend(q[31..0]);
```

例外： 无。

A.1.10 MODU.G （龙芯无符号字求模）



31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	MODU.G 0 1 1 1 1 0
6	5	5	5	5	6

格式： `MODU.G rd, rs, rt`

功能： 32 位无符号整数求模。

描述： 通用寄存器 rs 中 32 位值除以通用寄存器 rt 中 32 位值，这两个操作数都是无符号数。32 位剩余数保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：
```
q  ← rs[31..0] mod rt[31..0]; /* 无符号求模 */
rd ← sign_extend(q[31..0]);
```

例外： 无。

A.1.11 DMOD.G （龙芯双字求模）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	DMOD.G 0 1 1 1 0 1
6	5	5	5	5	6

格式： `DMOD.G rd, rs, rt`

功能： 64 位有符号整数求模。

描述： 通用寄存器 rs 中 64 位值除以通用寄存器 rt 中 64 位值，这两个操作数都是有符号数。64 位剩余数保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：
```
rd ← rs mod rt; /* 有符号求模 */
```

例外： 无。

A.1.12 DMODU.G （龙芯无符号双字求模）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	rs	rt	rd	0	DMODU.G 0 1 1 1 1 1
6	5	5	5	5	6

格式： `DMODU.G rd, rs, rt`

功能： 64 位无符号整数求模。

描述： 通用寄存器 rs 中 64 位值除以通用寄存器 rt 中 64 位值，这两个操作数都是无符号数。64 位剩余数保存在目的寄存器 rd 中。

任何情况下都不会产生算术异常。

操作：
```
rd ← rs mod rt; /* 无符号求模 */
```

例外： 无。

附录 B　龙芯新增浮点指令

B.1　单精度对指令

OP	OP(*fmt*= 22)	OP	OP(*fmt*= 22)
ADD	ADD.*ps*	C.*ult*	C.*ult*.*ps*
MADD	MADD.*ps*	C.*ole*	C.*ole*.*ps*
MSUB	MSUB.*ps*	C.*ule*	C.*ule*.*ps*
NMADD	NMADD.*ps*	C.*sf*	C.*sf*.*ps*
NMSUB	NMSUB.*ps*	C.*ngle*	C.*ngle*.*ps*
SUB	SUB.*ps*	C.*seq*	C.*seq*.*ps*
NEG	NEG.*ps*	C.*ugl*	C.*ugl*.*ps*
ABS	ABS.*ps*	C.*lt*	C.*lt*.*ps*
C.*f*	C.*f*.*ps*	C.*nge*	C.*nge*.*ps*
C.*un*	C.*un*.*ps*	C.*le*	C.*le*.*ps*
C.*eq*	C.*eq*.*ps*	C.*ngt*	C.*ngt*.*ps*
C.*ueq*	C.*ueq*.*ps*	MUL	MUL.*ps*
C.*olt*	C.*olt*.*ps*	MOV	MOV.*ps*

表 B.1: 龙芯单精度对指令

B.2　新增浮点指令

B.2.1　MADD.*fmt*（浮点乘加）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	fmt	ft	fs	fd	MADD 0 1 1 0 0 0
6	5	5	5	5	6

格式： `MADD.S fd, fs, ft`

`MADD.D fd, fs, ft`

功能： 浮点值的先乘后加。

描述： `fd ← (fs * ft) + fd;`

先将浮点寄存器 ft 中的值乘以浮点寄存器 fs 中的值，得到一个乘积。再把这个乘积加上浮点寄存器 fd 中的值，得到运算结果。这个运算指令结果有很高的精度，发生进位时的处理方式是按照 FCSR 的当前进位模式，结果保存进 fd 中。操作数和运算结果都是 *fmt* 格式。

操作：

```
vfd ← ValueFPR(fd, fmt);
vfs ← ValueFPR(fs, fmt);
vft ← ValueFPR(ft, fmt);
StoreFPR(fd, fmt, vfd + vfs * vft);
```

例外： 不可用协处理器例外
保留指令例外
浮点:

不精确例外	未实现操作例外	无效操作例外
上溢例外	下溢例外	

B.2.2 MSUB.*fmt*（浮点乘减）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
SPECIAL2 0 1 1 1 0 0	fmt	ft	fs	fd	MSUB 0 1 1 0 0 1
6	5	5	5	5	6

格式：

```
MSUB.S fd, fs, ft
MSUB.D fd, fs, ft
```

功能： 浮点值的先乘后减。

描述： `fd ← (fs * ft) - fd;`

先将浮点寄存器 ft 中的值乘以浮点寄存器 fs 中的值，得到一个乘积。再把这个乘积减去浮点寄存器 fd 中的值，得到运算结果。这个运算指令结果有很高的精度，发生进位时的处理方式是按照 FCSR 的当前进位模式，结果保存进 fd 中。操作数和运算结果都是 *fmt* 格式。

操作：

```
vfd ← ValueFPR(fd, fmt);
vfs ← ValueFPR(fs, fmt);
vft ← ValueFPR(ft, fmt);
StoreFPR(fd, fmt, (vfs * vft) - vfd);
```

例外： 不可用协处理器例外
保留指令例外
浮点:

不精确例外	未实现操作例外	无效操作例外
上溢例外	下溢例外	

B.2.3 NMADD.*fmt*（浮点乘加取负）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	fmt	ft	fs	fd	NMADD 0 1 1 0 1 0
6	5	5	5	5	6

格式： `NMADD.S fd, fs, ft`
`NMADD.D fd, fs, ft`

功能： 对先乘后加的结果取负。

描述： `fd ← -((fs * ft) + fd);`

先将浮点寄存器 ft 中的值乘以浮点寄存器 fs 中的值，得到一个乘积。再把这个乘积加上浮点寄存器 fd 中的值，再把和值取负得到运算结果。这个运算指令结果有很高的精度，发生进位时的处理方式是按照 FCSR 的当前进位模式，结果保存进 fd 中。操作数和运算结果都是 *fmt* 格式。

操作：

```
vfd ← ValueFPR(fd, fmt);
vfs ← ValueFPR(fs, fmt);
vft ← ValueFPR(ft, fmt);
StoreFPR(fd, fmt, -((vfs * vft) + vfd);
```

例外： 不可用协处理器例外
保留指令例外
浮点:
不精确例外　未实现操作例外　无效操作例外
上溢例外　下溢例外

B.2.4 NMSUB.*fmt*（浮点乘减取负）

31–26	25–21	20–16	15–11	10–6	5–0
SPECIAL2 0 1 1 1 0 0	fmt	ft	fs	fd	MSUB 0 1 1 0 0 1
6	5	5	5	5	6

格式： `NMSUB.S fd, fs, ft`
`NMSUB.D fd, fs, ft`

功能： 对先乘后减的结果取负。

描述： `fd ← -((fs * ft) - fd);`

先将浮点寄存器 ft 中的值乘以浮点寄存器 fs 中的值，得到一个乘积。再把这个乘积减去浮点寄存器 fd 中的值，再把差值取负得到运算结果。这个运算指令结果有很高的精度，发生进位时的处理方式是按照 FCSR 的当前进位模式，结果保存进 fd 中。操作数和运算结果都是 *fmt* 格式。

操作：

```
vfd ← ValueFPR(fd, fmt);
vfs ← ValueFPR(fs, fmt);
vft ← ValueFPR(ft, fmt);
StoreFPR(fd, fmt, -((vfs * vft) - vfd));
```

例外： 不可用协处理器例外
保留指令例外
浮点:
不精确例外 未实现操作例外 无效操作例外
上溢例外 下溢例外

附录 C　龙芯新增多媒体指令

龙芯的多媒体指令扩展通过增加新的指令和新的 64 位数据类型来提高多媒体及通信应用的表现。新增加的指令集没有增加任何新的操作模型或者需要操作系统维护的状态，保持了与现有龙芯平台上软硬件的兼容性。

现在多媒体、通信、以及图像软件大量地使用在较小的数据元素上进行多层重复操作的复杂算法（iterative algorithms）。龙芯的多媒体指令就是为了这个需求而产生的。比如，多数声音处理程序通常使用 16 位数据，而视频和图像软件则多采用 8 位数据。龙芯的多媒体指令采用了单指令多数据（single insruction multiple data，简称 SIMD）技术，用于提高软件并行处理多个数据元素的能力。

龙芯多媒体技术支持在 64 位上的字节，半字，字和双字元素的并行处理。多媒体指令为每 64 位数据引入了新的包（Packed）数据类型。包数据类型有如下几类：

- 字节包：字节是一个 8 位的数据结构，每个字节包是 8 个连续字节的集合；
- 半字包：半字是一个 16 位的数据结构，每个半字包是 4 个连续半字的集合；
- 字包：字是一个 32 位的数据结构，每个字包是 2 个连续字的集合；
- 双字包：双字是一个 64 位的数据结构，每个双字包就是一个 64 位的双字。

包内的单个数据元素都是整数类型，但可能有两种格式保存：无符号和有符号。对每个单独的元素，数据存放的规则是：较低位存放数据低位，较高位存放数据高位。每个数据元素也有它对应的最低位（leset significant bit，简称 LSB），和最高位（most significant bit，简称 MSB）。图 C.1 给出了这四种无符号包的数据格式；有符号包有类似的格式，不同的是每个数据元素的最高位被作为符号位。

在龙芯 GS464 处理器核上，多媒体指令的编码是通过对 MIPS 指令的协处理器 2 （CP2）指令进行自定义而实现的，所以执行这些指令时要求协处理器 2 可用，即 CP0 的 Status 寄存器的 CU[2] 必须使能，其中浮点相关指令还要求协处理器 1 可用。

C.1　多媒体指令特性

C.1.1　饱和模式和截断模式

当执行整数运算时，尤其是当数据元素的位数比较少的时候，运算结果很容易超出可以表达的范围而导致不能保存。例如，当有符号半字整数运算，可能会出现结果溢出，即最后结果大于 16 位可以存储的范围。当这种情况发生时，龙芯的多媒体指令提供了三种处理超出范围的方法。它们分别是：

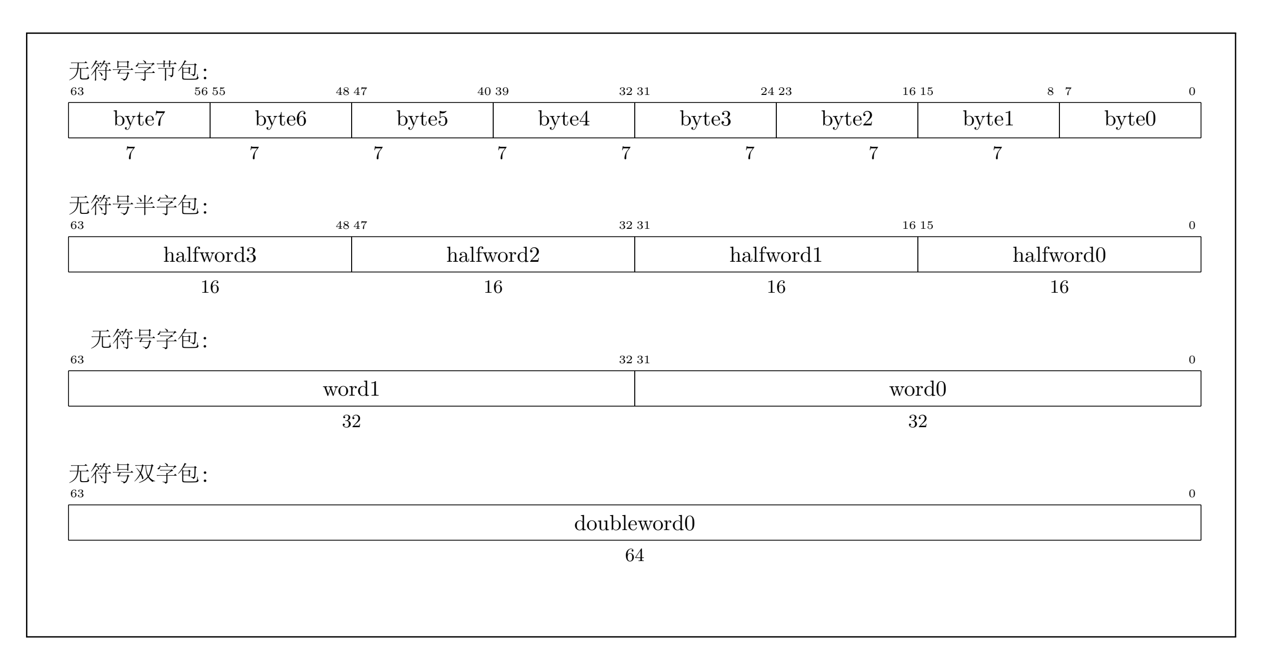


图 C.1: 无符号数据包格式

- 截断模式

  截断模式是简单地将超出范围的结果截断，即溢出位都被忽略，只有有效的低位被保留，并存入目的。使用截断模式的应用程序应当控制操作数的范围，以防止超出范围的现象发生。如果操作数的范围不加以控制，截断模式可能产生较大的误差。例如，两个大的有符号数相加，如果产生正向的溢出。在截断模式下，结果可能是负数。

- 有符号饱和模式

  有符号饱和模式是在溢出发生时，尽可能的在有符号数的范围去表达超出范围的结果。例如，当有符号半字整数运算时，如果正向的溢出发生，那么结果将会被“饱和”到 7FFFH — 因为这是在 16 位的范围内能表达的最大整数。同理，如果，负向溢出发生，那么结果为 8000H，即最小的半字整数。

- 无符号饱和模式

  无符号饱和模式与有符号饱和模式类似，它的思想是在溢出发生时，在无符号数的范围内尽量地表达溢出的无符号数。因此，当上溢发生时，无符号字节整数的结果为 FFH，而下溢的结果会被饱和为 00H。其他数据类型同理可得。

一般而言，饱和模式在算术溢出的情况下，会给出更自然的结果。比如，在色彩的计算中，饱和模式将会使得颜色结果为纯黑或纯白色（最大或最小值）而不会出现颜色反转的情况。当不可能检查源操作数范围时，它还同时可以避免计算由于截断的原因出现奇怪的结果。

龙芯的多媒体指令不会因为计算中出现上溢或下溢的情况而触发例外。

C.1.2 多媒体指令列表及格式

多媒体指令的操作对象有以下几种不同情况：

- 数据类型：字节包，半字包，字包，或者双字包？

- 符号位：有符号或无符号数？
- 溢出控制：饱和或截断模式？

fmt	FUNC			
	ADD 000000	SUB 000001	MUL 000010	DIV 000011
24	PADDSH	PSUBSH	PSHUFH	PUNPCKLHW
25	PADDUSH	PSUBUSH	PACKSSWH	PUNPCKHHW
26	PADDH	PSUBH	PACKSSHB	PUNPCKLBH
27	PADDW	PSUBW	PACKUSHB	PUNPCKHBH
28	PADDSB	PSUBSB	PXOR	PINSRH_0
29	PADDUSB	PSUBUSB	PNOR	PINSRH_1
30	PADDB	PSUBB	PAND	PINSRH_2
31	PADDD	PSUBD	PANDN	PINSRH_3
fmt	ROUND.L 001000	TRUNC.L 001001	CEIL.L 001010	FLOOR.L 001011
24	PAVGH	PCMPEQW	PSLLW	PSRLW
25	PAVGB	PCMPGTW	PSLLH	PSRLH
26	PMAXSH	PCMPEQH	PMULLH	PSRAW
27	PMINSH	PCMPGTH	PMULHH	PSRAH
28	PMAXUB	PCMPEQB	PMULUW	PUNPCKLWD
29	PMINUB	PCMPGTB	PMULHUH	PUNPCKHWD
fmt	ROUND.W 001000	TRUNC.W 001001	CEIL.W 001010	FLOOR.W 001011
24	PADDU	PSUBU	PSLL	PSRL
25	POR	PASUBUB	PDSLL	PDSRL
26	PADD	PSUB	PEXTRH	PSRA
27	PDADD	PDSUB	PMADDHW	PDSRA
28	PSEQU	PSLTU	PSLEU	BIADD
29	PSEQ	PSLT	PSLE	PMOVMSKB

表 C.1: 龙芯多媒体指令集

表 C.1 列出了所有龙芯 GS464 新增的多媒体指令。通常，多媒体指令采用如下的格式：

- 前缀：P 表示指令操作包数据类型；
- 中段：具体指令，如 ADD，CMP，或 XOR 等；
- 后缀
  - US：无符号，饱和；
  - S: 有符号，饱和；
  - B，H，W，D：数据类型 — 分别表示字节包（byte），半字包（halfword），字包（word），或双字包（doubleword）。

如果一条指令有不同的输入和输出结果类型时，那么它将会有两个数据类型的后缀。比如说，转换指令将一种包类型数据转换为另一种包数据类型，那么它有两个数据类型后缀：第一个是输入数据类型，而第二个是输出数据类型。以下是一条龙芯多媒体指令的助记格式的实例：

```
PADDUSW （无符号字整数饱和模式加）
P   = 包数据
ADD = 指令操作
US  = 无符号饱和模式
W   = 字
```

C.2 多媒体指令详解

在以下的各小节中，将逐个对每个指令进行详细的介绍。

C.2.1 PACKSSHB/PACKSSWH （有符号数打包）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PACKSSHB 1 1 0 1 0	ft	fs	fd	MUL 0 0 0 0 1 0
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PACKSSWH 1 1 0 0 1	ft	fs	fd	MUL 0 0 0 0 1 0
6	5	5	5	5	6

格式：

```
PACK.SSHB fd,fs,ft
PACK.SSWH fd,fs,ft
```

功能： 饱和模式有符号数打包：将包结构转换为较窄的格式。

描述： PACKSSHB 将有符号半字包转换用饱和模式转换为有符号字节包，而 PACKSSWH 将有符号字包转换用饱和模式转换为有符号半字包。共同特点是使用饱和条件转换较宽的数据类型到较窄的数据类型。

PACKSSHB 的具体操作是将“第一个操作数的 4 个有符号整数半字和第二个操作数的 4 个有符号半字整数”，共 8 个有符号整数半字，转换为 8 个有符号字节整数，并将结果存入目标操作数。转换的过程使用饱和模式：在转换过程中，如果一个有符号半字整数的值超出一个有符号字节的整数范围（大于 EFH 或小于 80H），则生成的的有符号字节整数分别为 EFH 或 80H。

PACKSSWH 的操作类似，不同的是操作对象是有符号字整数，并生成有符号半字整数。

操作：

```
PACKSSHB
fd[ 7..0 ] ← SaturateSignedHalfwordToSignedByte fs[15..0 ];
fd[15..8 ] ← SaturateSignedHalfwordToSignedByte fs[31..16];
fd[23..16] ← SaturateSignedHalfwordToSignedByte fs[47..32];
fd[31..24] ← SaturateSignedHalfwordToSignedByte fs[63..48];
fd[39..32] ← SaturateSignedHalfwordToSignedByte ft[15..0 ];
fd[47..0 ] ← SaturateSignedHalfwordToSignedByte ft[31..16];
fd[55..48] ← SaturateSignedHalfwordToSignedByte ft[47..32];
fd[63..56] ← SaturateSignedHalfwordToSignedByte ft[63..48];

PACKSSWH
fd[15..0 ] ← SaturateSignedWordToSignedHalfWord fs[31..0 ];
fd[31..16] ← SaturateSignedWordToSignedHalfWord fs[63..32];
fd[47..32] ← SaturateSignedWordToSignedHalfWord ft[31..0 ];
fd[63..48] ← SaturateSignedWordToSignedHalfWord ft[63..32];
```

例外： 无。

C.2.2 PACKUSHB （无符号数打包）

31–26	25–21	20–16	15–11	10–6	5–0
COP2	PACKUSHB	ft	fs	fd	MUL
0 1 0 0 1 0	1 1 0 1 1				0 0 0 0 1 0
6	5	5	5	5	6

格式： PACKUSHB fd,fs,ft

功能： 饱和模式无符号数打包

描述： PACKUSHB 将有符号半字包转换用饱和模式转换为无符号字节包。具体的操作是将“第一个操作数的 4 个有符号整数半字和第二个操作数的 4 个有符号整数半字”，共 8 个有符号整数半字，转换为 8 个无符号字节整数，并将结果存入目标操作数。转换的过程使用饱和模式：在转换过程中，如果一个有符号半字整数的值超出一个无符号字节的整数范围（大于 FFH 或小于 00H），则生成的的无符号整数字节分别为 FFH 或 00H。

操作：

```
PACKUSHB
fd[ 7..0 ] ← SaturateSignedHalfwordToUnsignedByte fs[15..0 ];
fd[15..8 ] ← SaturateSignedHalfwordToUnsignedByte fs[31..16];
fd[23..16] ← SaturateSignedHalfwordToUnsignedByte fs[47..32];
fd[31..24] ← SaturateSignedHalfwordToUnsignedByte fs[63..48];
fd[39..32] ← SaturateSignedHalfwordToUnsignedByte ft[15..0 ];
fd[47..0 ] ← SaturateSignedHalfwordToUnsignedByte ft[31..16];
fd[55..48] ← SaturateSignedHalfwordToUnsignedByte ft[47..32];
fd[63..56] ← SaturateSignedHalfwordToUnsignedByte ft[63..48];
```

例外： 无。

C.2.3 PADDB/PADDH/PADDW/PADDD （整数包加）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PADDB 1 1 1 1 0	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PADDH 1 1 0 1 0	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PADDW 1 1 0 1 1	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PADDD 1 1 1 1 1	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

格式：

```
PADDB fd,fs,ft
PADDH fd,fs,ft
PADDW fd,fs,ft
PADDD fd,fs,ft
```

功能： 使用截断模式执行各种整数包类型相加运算。

描述： 这四条指令用于执行各种整数包加法操作：它们分别将第一个和第二个操作寄存器的对应位置的各个操作数（依照指令给定的数据类型）相加，并将结果存储在目标操作数中。溢出处理为截断模式。比如说，PADDB 指令的具体操作是将第一个和第二个操作寄存器对应的 8 个字节整数相加。当任何一个加法结果太大，即在第 9 位溢出时，只有低 8 位结果将被写入到目标操作数（进位被忽略）。其他三个指令与 PADDB 类似。

注意，这四条包加法指令既可以应用于无符号，也可以应用于（采用 2 的补码表示法的）有符号整数包。为了避免溢出或进位情况的发生，软件必须控制操作数的范围。

操作：

```
PADDB
fd[ 7..0 ] ← fs[ 7..0 ] + ft[ 7..0 ];
* repeat add operation for 2nd through 7th byte *;
fd[63..56] ← fs[63..56] + ft[63..56];

PADDH
fd[15..0 ] ← fs[15..0 ] + ft[15..0 ];
* repeat add operation for 2nd and 3th halfword *;
fd[63..48] ← fs[63..48] + ft[63..48];

PADDW
fd[31..0 ] ← fs[31..0 ] + ft[31..0 ];
fd[63..32] ← fs[63..32] + ft[63..32];

PADDD
fd[63..0 ] ← fs[63..0 ] + ft[63..0 ];
```

例外： 无。

C.2.4 PADDSB/PADDSH （有符号整数包加）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PADDSB 1 1 1 0 0	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PADDSH 1 1 0 0 0	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

格式：

```
PADDSB fd,fs,ft
PADDSH fd,fs,ft
```

功能： 有符号整数包加

描述： 这两条指令用于执行各种有符号整数包的加法操作：它们分别将第一个和第二个操作寄存器的对应位置的各个操作数（依照指令给定的数据类型）相加，并将结果存储在目标操作数中。溢出处理采用有符号饱和模式。这些指令的操作数皆为 64 位。

PADDSB 指令的具体操作是将第一个和第二个操作寄存器对应的 8 个有符号字节整数相加。当任何一个加法结果太大或太小，超出了一个有符号字节的整数范围（即大于 7FH 或少于 80H），则对应的饱和值 7FH 或 80H，将被写入目的操作数。相应的，PADDSH 指令操作有符号半字整数。当一个半字相加的结果超出了有符号半字整数范围时（即大于 7FFFH 或小于 8000H），则 7FFFH 或 8000H 的作为饱和值，将写入目标操作数。

操作：

```
PADDSB
fd[ 7..0 ] ← SaturateToSignedByte(fs[ 7..0 ] + ft[ 7..0 ]);
* repeat add operation for 2nd through 7th bytes *;
fd[63..56] ← SaturateToSignedByte(fs[63..56] + ft[63..56]);

PADDSH
fd[15..0 ] ← SaturateToSignedHalfword(fs[15..0 ] + ft[15..0 ]);
* repeat add operation for 2nd and 7th halfwords *;
fd[63..48] ← SaturateToSignedHalfword(fs[63..48] + ft[63..48]);
```

例外： 无。

C.2.5 PADDUSB/PADDUSH （无符号整数包加）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PADDUSB 1 1 1 0 1	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PADDUSH 1 1 0 0 1	ft	fs	fd	ADD 0 0 0 0 0 0
6	5	5	5	5	6

格式：

```
PADDUSB fd,fs,ft
PADDUSH fd,fs,ft
```

功能： 无符号整数包加法运算。

描述： 这两条指令用于执行各种无符号整数包加操作：它们分别将第一个和第二个操作寄存器的对应位置的各个操作数（依照指令给定的数据类型）相加，并将结果存储在目标操作数中。溢出处理采用无符号饱和模式。

PADDUSB 指令的具体操作是将第一个和第二个操作寄存器对应的 8 个无符号字节整数相加。当任何一个加法结果太大，超出了一个无符号字节的整数范围（即大于 FFH)，则 FFH 将作为饱和值写入目的操作数。相应的，PADDUSH 指令操作无符号半字整数。当一个半字相加的结果超出了无符号半字整数范围时（即大于 FFFFH)，则 FFFFH 作为饱和值将写入目标操作数。

操作：

```
PADDUSB
fd[ 7..0 ] ← SaturateToUnsignedByte(fs[ 7..0 ] + ft[ 7..0 ]);
* repeat add operation for 2nd through 7th bytes *;
fd[63..56] ← SaturateToUnsignedByte(fs[63..56] + ft[63..56]);

PADDUSH
fd[15..0 ] ← SaturateToUnsignedHalfword(fs[15..0 ] + ft[15..0 ]);
* repeat add operation for 2nd and 3rd halfwords *;
fd[63..48] ← SaturateToUnsignedHalfword(fs[63..48] + ft[63..48]);
```

例外： 无。

C.2.6 PANDN （逐位逻辑与非）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PANDN 1 1 1 1 1	ft	fs	fd	MUL 0 0 0 0 1 0
6	5	5	5	5	6

格式： `PANDN fd,fs,ft`

功能： 逐位逻辑与非操作

描述： 该指令执行逐位逻辑与非操作。具体操作为，首先对第一个操作数进行按位非逻辑运算，然后将结果和第二个操作数进行按位逻辑与运算，并将结果写入目标操作数。所有源和目的操作数都是 64 位寄存器。

操作：

```
PANDN
fd←(NOT fs) AND ft;
```

例外： 无。

C.2.7 PAVGB/PAVGH （无符号整数包求平均）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PAVGB 1 1 0 0 1	ft	fs	fd	ROUND.L 0 0 1 0 0 0
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PAVGH 1 1 0 0 0	ft	fs	fd	ROUND.L 0 0 1 0 0 0
6	5	5	5	5	6

格式： PAVGB fd,fs,ft
PAVGH fd,fs,ft

功能： 包整数求平均值

描述： 这两条指令执行无符号整数包求平均操作：对第一个操作数和第二个操作数对应位置的无符号整数求平均，并将结果存储在目标操作数中。在操作中，当第一个操作数中的元素和第二个操作数中的对应元素相加时，其和还被加 1，然后结果右移一位（除 2 操作）。注意，加 1 的操作的实质效果是使得求平均结果为四舍五入。所有的源和目的操作数皆是 64 位寄存器。

PAVGB 指令用于无符号字节整数包，而 PAVGH 用于无符号半字整数包。

操作：

```
PAVGB
* temp sum before shifting is 9 bits *
ft[ 7..0 ] ← (fs[ 7..0 ] + ft[ 7..0 ] + 1) >> 1;
* 对第二到第七个字节重复以上操作 *;
ft[63..56] ← (fs[63..56] + ft[63..56] + 1) >> 1;

PAVGH
* temp sum before shifting is 17 bits *
ft[ 1..0 ] ← (fs[15..0 ] + ft[15..0 ] + 1) >> 1;
* repeat operation performed for halfwords 2 and 3 *;
ft[63..48] ← (fs[63..48] + ft[63..48] + 1) >> 1;
```

例外： 无。

C.2.8 PCMPEQB/PCMPEQH/PCMPEQW （整数包相等比较）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PCMPEQB 1 1 1 0 0	ft	fs	fd	TRUNC.L 0 0 1 0 0 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PCMPEQH 1 1 0 1 0	ft	fs	fd	TRUNC.L 0 0 1 0 0 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PCMPEQW 1 1 0 0 0	ft	fs	fd	TRUNC.L 0 0 1 0 0 1
6	5	5	5	5	6

格式： PCMPEQB fd,fs,ft
PCMPEQH fd,fs,ft
PCMPEQW fd,fs,ft

功能： 字节整数包的各元素是否相等

描述： 这三条指令执行对第一个和第二个操作数中的整数包的各元素逐个进行相等比较操作：如果一对数据元素相等，则目标操作数的相应数据元素所有位皆设为 1，否则全置为 0。PCMPEQB 指令用于比较第一和第二操作数的字节元素；PCMPEQH 指令用于比较第一和第二操作数的半字元素；PCMPEQW 指令用于比较第一和第二操作数的字元素。

操作：

```
PCMPEQB
fd[ 7..0 ] ← fs[ 7..0 ] == ft[ 7..0 ] ? FFH : 0;
* continue comparison of 2nd through 7th bytes in fs and ft *
fd[63..56] ← fs[63..56] == ft[63..56] ? FFH : 0;

PCMPEQH
fd[15..0 ] ← fs[15..0 ] == ft[15..0 ] ? FFFFH : 0;
* continue comparison of 2nd and 3rd halfwords in fs and ft *
fd[63..48] ← fs[63..48] == ft[63..48] ? FFFFH : 0;

PCMPEQW
fd[31..0 ] ← fs[31..0 ] == ft[31..0 ] ? FFFFFFFFH : 0;
fd[63..32] ← fs[63..32] == ft[63..32] ? FFFFFFFFH : 0;
```

例外： 无。

C.2.9 PEXTRH （半字抽取操作）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PEXTRH 1 1 0 1 0	ft	fs	fd	CELL.W 0 0 1 1 1 0
6	5	5	5	5	6

格式： `PEXTRH fd,fs,ft`

功能： 抽取指定的半字到目的寄存器

描述： 该指令复制由第一个操作数的半字到目的操作数，该半字的位置为第二个操作数的低两位决定指定的。目的操作数的三个高位半字被清零。

操作：

```
PEXTRH
SEL ← ft AND 3H;
TEMP← fs >> (SEL * 16);
fd[15..0 ] ← TEMP[15..0 ];
fd[63..16] ← 00000000H;
```

例外： 无。

C.2.10 PINSRH （半字插入操作）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PINSRH_0 1 1 1 0 0	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PINSRH_1 1 1 1 0 1	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PINSRH_2 1 1 1 1 0	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

31..26	25..21	20..16	15..11	10..6	5..0
COP2 0 1 0 0 1 0	PINSRH_3 1 1 1 1 1	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

格式：

```
PINSRH_0 fd,fs,ft
PINSRH_1 fd,fs,ft
PINSRH_2 fd,fs,ft
PINSRH_3 fd,fs,ft
```

功能：

描述： 复制第二个操作数的低位半字，插入到第一个操作数的指定位置，第一个操作数的其他半字保持不变，并将结果存入目的寄存器。

操作：

```
PINSRH_0
MASK← 000000000000FFFFH;
fd ← (fs AND NOT MASK) OR ((ft << (0 * 16)) AND MASK);
PINSRH_1
MASK← 00000000FFFF0000H;
fd ← (fs AND NOT MASK) OR ((ft << (1 * 16)) AND MASK);
PINSRH_2
MASK← 0000FFFF00000000H;
fd ← (fs AND NOT MASK) OR ((ft << (2 * 16)) AND MASK);
PINSRH_3
MASK← FFFF000000000000H;
fd ← (fs AND NOT MASK) OR ((ft << (3 * 16)) AND MASK);
```

例外： 无。

C.2.11 PMADDHW （有符号整数包 -半字到字 -乘加运算）

31..26	25..21	20..16	15..11	10..6	5..0
COP2 0 1 0 0 1 0	PMADDHW 1 1 0 1 1	ft	fs	fd	CELL.W 0 0 1 1 1 0
6	5	5	5	5	6

格式：

```
PMADDHW fd,fs,ft
```

功能： 有符号半字整数包先乘后加，结果为字整数包类型

描述： 该指令的具体操作为：首先将第一个操作数的四个有符号半字相乘分别与第二个操作数的相应有符号半字相乘，临时结果是四个有符号字。前两个字和后两个字分别后相加，结果存储在目的操作数。因为字整数被用来存储半字乘加的结果，一般而言，溢出不会发生。PMADDHW 指令的溢出截断只会在一种情形下生效：当在一个操作组的两对半字，都是 8000H 时。在这种情况下，乘加的结果将会被截断为 80000000H。

操作：

```
PMADDHW
fd[31..0 ] ← (fs[15..0 ] * ft[15..0 ]) + (fs[31..16] * ft[31..16]);
fd[63..32] ← (fs[47..32] * ft[47..32]) + (fs[63..48] * ft[63..48]);
```

例外： 无。

C.2.12 PMAXSH/PMINSH （有符号半字整数包极值运算）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PMAXSH 1 1 0 1 0	ft	fs	fd	ROUND.L 0 0 1 0 0 0
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PMINSH 1 1 0 1 1	ft	fs	fd	ROUND.L 0 0 1 0 0 0
6	5	5	5	5	6

格式：
```
PMAXSH fd,fs,ft
PMINSH fd,fs,ft
```

功能： 对两个有符号半字整数包的四对半字分别求极值。

描述： 这两条指令对第一个操作数和第二个操作数的四对有符号半字分别求最大最小值，并将结果存入目的寄存器。

操作：
```
PMAXSH
fd[15..0 ] ← max(fs[15..0 ], ft[15..0 ]); /* 有符号比较 */
* 对第二和第三个半字重复以上操作
fd[63..48] ← max(fs[63..48], ft[63..48]);

PMINSH
fd[15..0 ] ← min(fs[15..0 ], ft[15..0 ]); /* 有符号比较 */
* 对第二和第三个半字重复以上操作
fd[63..48] ← min(fs[63..48], ft[63..48]);
```

例外： 无。

C.2.13 PMAXUB/PMINUB （无符号字节整数包极值运算）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PMAXUB 1 1 1 0 0	ft	fs	fd	ROUND.L 0 0 1 0 0 0
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PMINUB 1 1 1 0 1	ft	fs	fd	ROUND.L 0 0 1 0 0 0
6	5	5	5	5	6

格式：
```
PMAXUB fd,fs,ft
PMINUB fd,fs,ft
```

功能： 对两个无符号字节整数包的八对字节整数分别求极值。

描述： 这两条指令对第一个操作数和第二个操作数的八对无符号字节整数分别求最大最小值，并将结果存入目的寄存器。

操作：

```
PMAXUB
fd[ 7..0 ] ← max(fs[ 7..0 ], ft[ 7..0 ]); /* 无符号比较 */
* 对第二到第七个字节重复以上操作 *
fd[63..56] ← max(fs[63..56], ft[63..56]);

PINXUB
fd[ 7..0 ] ← min(fs[ 7..0 ], ft[ 7..0 ]); /* 无符号比较 */
* 对第二到第七个字节重复以上操作 *
fd[63..56] ← min(fs[63..56], ft[63..56]);
```

例外： 无。

C.2.14 PMOVMSKB （字节掩码移动操作）

Remark: two operands!

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PMOVMSKB 1 1 1 0 1	ft	fs	fd	FLOOR.W 0 0 1 1 1 1
6	5	5	5	5	6

格式： `PMOVMSKB fd,fs`

功能： 将字节整数包的 8 个字节掩码归并到一个字节上。

描述： 该指令是将第一个操作数（一个 64 位的字节整数包）的八个字节的最高位（即掩码位）归并且存储在目标操作数的低字节中。

操作：

```
PMOVMSKB
fd[0] ← fs[ 7];
fd[1] ← fs[15];
* 对第三到第七个字节重复以上操作 *
fd[7] ← fs[63];
fd[63..8 ] ← 00000000000000H;
```

例外： 无。

C.2.15 PMULHUH （无符号半字整数包乘：高位结果）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PMULHUH 1 1 1 0 1	ft	fs	fd	CELL.L 0 0 1 0 1 0
6	5	5	5	5	6

格式： `PMULHUH fd,fs,ft`

功能： 无符号半字整数包的乘法运算，并存储高位结果。

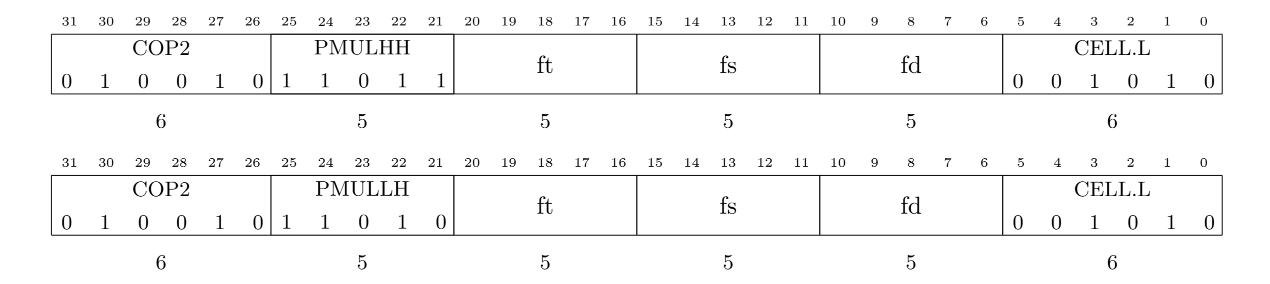
描述： 该指令执行无符号半字整数包的乘法运算，并将四个乘法结果的高半字结果存储到目的寄存器中。

操作：

```
PMULHUH
fd[15..0 ] ← (fs[15..0 ] * ft[15..0 ])[31..16]; /* 无符号乘 */
fd[31..16] ← (fs[31..16] * ft[31..16])[31..16];
fd[47..32] ← (fs[47..32] * ft[47..32])[31..16];
fd[63..48] ← (fs[63..48] * ft[63..48])[31..16];
```

例外： 无。

C.2.16 PMULHH/PMULLH （有符号半字整数包乘：高位、低位结果）



格式：
```
PMULHH fd,fs,ft
PMULLH fd,fs,ft
```

功能： 有符号半字整数包的乘法运算，并分别存储高位、低位结果。

描述： 这两条指令执行无符号半字整数包的乘法运算，并分别将四个乘法结果的高半字或者低半字结果存储到目的寄存器中。

操作：
```
PMULHH
fd[15..0 ] ← (fs[15..0 ] * ft[15..0 ])[31..16]; /* 有符号乘 */
fd[31..16] ← (fs[31..16] * ft[31..16])[31..16];
fd[47..32] ← (fs[47..32] * ft[47..32])[31..16];
fd[63..48] ← (fs[63..48] * ft[63..48])[31..16];

PMULLH
fd[15..0 ] ← (fs[15..0 ] * ft[15..0 ])[15..0 ]; /* 有符号乘 */
fd[31..16] ← (fs[31..16] * ft[31..16])[15..0 ];
fd[47..32] ← (fs[47..32] * ft[47..32])[15..0 ];
fd[63..48] ← (fs[63..48] * ft[63..48])[15..0 ];
```

例外： 无。

C.2.17 PMULUW （无符号字整数包乘）

31–26	25–21	20–16	15–11	10–6	5–0
COP2	PMULUW	ft	fs	fd	CELL.L
0 1 0 0 1 0	1 1 1 0 0				0 0 1 0 1 0
6	5	5	5	5	6

格式：
```
PMULUW fd,fs,ft
```

功能： 执行无符号字整数包乘法运算，溢出采用截断模式。

描述： 该指令执行无符号字整数包乘法运算，具体操作为将第一个操作数的低字整数与第二个操作数低字整数相乘，并将无符号整数结果存储 64 的目的操作数。源操作数和结果都是无符号整数。如果结果太大，在 64 位上溢出，则将会被截断，只有低 64 位写入到目标中（即进位被忽略）。

操作：
```
PMULUW
fd[63..0] ← fs[31..0] * ft[31..0];
```

例外： 无。

C.2.18 PASUBUB （无符号字节包绝对差值运算）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2	PASUBUB	ft	fs	fd	TRUNC.W
0 1 0 0 1 0	1 1 0 0 1				0 0 1 1 0 1
6	5	5	5	5	6

格式： `PASUBUB fd,fs,ft`

功能： 计算无符号字节整数包的对应元素的绝对差值。

描述： PSADBH 指令计算两个无符号字节整数包的八对字节整数的绝对差值，即差的绝对值，物理意义一般就是距离。

操作：

```
PASUBUB
fd[ 7..0 ] ← ABS(fs[ 7..0 ] - ft[ 7..0 ]);
* repeat operation for bytes 2 through 6 *
fd[63..56] ← ABS(fs[63..56] - ft[63..56]);
```

例外： 无。

C.2.19 BIADD （无符号字节包内部和）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2	BIADD	ft	fs	fd	FLOOR.W
0 1 0 0 1 0	1 1 1 0 0				0 0 1 1 1 1
6	5	5	5	5	6

格式： `BIADD fd,fs`

功能： 计算无符号字节包内所有字节整数的和，结果为一个半字整数。

描述： PSADBH 指令计算第一个操作数（一个无符号字节整数包）的八个字节整数的和。结果是一个无符号的半字整数，并存储在目的操作数的低字位置，目标操作数的其余字节都将被清零。

操作：

```
BIADD
fd[15..0 ] ← SUM(fs[7..0], fs[15..8], ..., fs[63..56]);
fd[63..16] ← 000000000000H;
```

例外： 无。

C.2.20 PSHUFH （半字包换位操作）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2	PSHUFH	ft	fs	fd	MUL
0 1 0 0 1 0	1 1 0 0 0				0 0 0 0 1 0
6	5	5	5	5	6

格式： `PSHUFH fd,fs,ft`

功能： 将半字包按指定的位置换位并复制到目的寄存器。

描述： 该指令的具体操作是将第一个操作数的各个半字按第二个操作数给出的位置插入到目的操作数中去。第二个操作数对每个半字位置用两位（2 bits）来编码。具体编码为 —— 00_2: 半字$_0$; 01_2: 半字$_1$; 10_2: 半字$_2$; 11_2: 半字$_3$;。注意，该指令允许在一个半字被复制到目的操作数的多个位置。

操作：

```
PSHUFH
fd[15..0 ] ← (fs >> (ft[1..0] * 16))[15..0]
fd[31..16] ← (fs >> (ft[3..2] * 16))[15..0]
fd[47..32] ← (fs >> (ft[5..4] * 16))[15..0]
fd[63..48] ← (fs >> (ft[7..6] * 16))[15..0]
```

例外： 无。

C.2.21 PSLLH/PSLLW （整数包逻辑左移位）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSLLH 1 1 0 0 1	ft	fs	fd	CELL.L 0 0 1 0 1 0
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSLLW 1 1 0 0 0	ft	fs	fd	CELL.L 0 0 1 0 1 0
6	5	5	5	5	6

格式：

```
PSLLH fd,fs,ft
PSLLW fd,fs,ft
```

功能： 半字或字整数包的逻辑左移位运算。

描述： 这两条指令分别对第一个操作数中的半字或字元素进行逻辑左移位运算，并将结果存储在目的寄存器中。移动的位数由第二个操作数的低 7 位指定。移位采用逻辑移位方式，即当数据元素被左移时，空出的低序位被零填充。如果移位值大于 15（半字），31（字），则移位结果为 0。

操作：

```
PSLLH
IF (ft[6..0 ] > 15)
THEN
   fd[64..0 ] ← 0000000000000000H
ELSE
   fd[15..0 ] ← ZeroExtend(fs[15..0 ] << ft[6..0]);
   * repeat shift operation for 2nd and 3rd words *;
   fd[63..48] ← ZeroExtend(fs[63..48] << ft[6..0]);
FI;

PSLLW
IF (ft[6..0 ] > 31)
THEN
   fd[64..0 ] ← 0000000000000000H
ELSE
   fd[31..0 ] ← ZeroExtend(fs[31..0 ] << ft[6..0]);
   fd[63..32] ← ZeroExtend(fs[63..32] << ft[6..0]);
FI;
```

例外： 无。

C.2.22 PSRAH/PSRAW （整数包算数右移位）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSRAH 1 1 0 1 1	ft	fs	fd	FLOOR.L 0 0 1 0 1 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSRAW 1 1 0 1 0	ft	fs	fd	FLOOR.L 0 0 1 0 1 1
6	5	5	5	5	6

格式：

```
PSRAH fd,fs,ft
PSRAW fd,fs,ft
```

功能： 半字或字整数包的算数右移位运算。

描述： 这两条指令分别对第一个操作数中的半字或字元素进行算数右移位运算，并将结果存储在目的寄存器中。移动的位数由第二个操作数的低 7 位指定。移位采用算数移位方式，即当数据元素被右移时，空出的高序位由初始的符号位的值填充。如果移位值大于 15（半字），31 （字），则移位结果为各位都是了元素符号位的初始值。

操作：

```
PSRAH
IF (ft[6..0] > 16) THEN ft[6..0] ← 16;
fd[15..0 ] ← SignExtend(fs[15..0 ] >> ft[6..0]);
* repeat shift operation for 2nd and 3rd halfwords *;
fd[63..48] ← SignExtend(fs[63..48] >> ft[6..0]);

PSRAW
IF (ft[6..0 ] > 31) THEN ft[6..0] ← 32;
fd[31..0 ] ← SignExtend(fs[31..0 ] >> ft[6..0]);
fd[63..32] ← SignExtend(fs[63..32] >> ft[6..0]);
```

例外： 无。

C.2.23 PSRLH/PSRLW （整数包逻辑右移位）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSRLH 1 1 0 0 1	ft	fs	fd	FLOOR.L 0 0 1 0 1 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSRLW 1 1 0 0 0	ft	fs	fd	FLOOR.L 0 0 1 0 1 1
6	5	5	5	5	6

格式：

```
PSRLH fd,fs,ft
PSRLW fd,fs,ft
```

功能： 半字或字整数包的逻辑右移位运算。

描述： 这两条指令分别对第一个操作数中的半字或字元素进行逻辑右移位运算，并将结果存储在目的寄存器中。移动的位数由第二个操作数的低 7 位指定。移位采用逻辑移位方式，即当数据元素被右移时，空出的高序位将被零填充。如果移位值大于 15（半字），31 （字），则移位结果为 0。

操作：

```
PSRLH
IF (ft[6..0] > 15)
THEN
    fd[64..0 ] ← 0000000000000000H
ELSE
    fd[15..0 ] ← ZeroExtend(fs[15..0 ] >> ft[6..0]);
    * repeat shift operation for 2nd and 3rd halfwords *;
    fd[63..48] ← ZeroExtend(fs[63..48] >> ft[6..0]);
FI;

PSRLW
IF (ft[6..0] > 31)
THEN
    fd[64..0 ] ← 0000000000000000H
ELSE
    fd[31..0 ] ← ZeroExtend(fs[31..0 ] >> ft[6..0]);
    fd[63..32] ← ZeroExtend(fs[63..32] >> ft[6..0]);
FI;
```

例外： 无。

C.2.24 PSUBB/PSUBH/PSUBW/PSUBD （整数包减）

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PSUBB 1 1 1 1 0	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PSUBH 1 1 0 1 0	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PSUBW 1 1 0 1 1	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

31 30 29 28 27 26	25 24 23 22 21	20 19 18 17 16	15 14 13 12 11	10 9 8 7 6	5 4 3 2 1 0
COP2 0 1 0 0 1 0	PSUBD 1 1 1 1 1	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

格式：

```
PSUBB fd,fs,ft
PSUBH fd,fs,ft
PSUBW fd,fs,ft
PSUBD fd,fs,ft
```

功能： 使用截断模式执行各种类型整数包相减运算。

描述： 这四条指令用于执行各种整数包的减法操作：它们分别将第一个和第二个操作寄存器的对应位置的各个操作数（依照指令给定的数据类型）相减，并将结果存储在目标操作数中。溢出处理为截断模式。比如说，PSUBB 指令的具体操作是将第一个和第二个操作寄存器对应的 8 个字节整数相减。当任何一个减法结果太大，即在第 9 位溢出时，只有低 8 位结果将被写入到目标操作数（进位被忽略）。其他三个指令与 PSUBB 类似。

注意，这四条包减法指令既可以应用于无符号，也可以应用于（采用 2 的补码表示法的）有符号整数包。为了避免溢出或进位情况的发生，软件必须控制操作数的范围。

操作：

```
PSUBB
fd[ 7..0 ] ← fs[ 7..0 ] - ft[ 7..0 ];
* repeat subtract operation for 2nd through 7th byte *;
fd[63..56] ← fs[63..56] - ft[63..56];

PSUBH
fd[15..0 ] ← fs[15..0 ] - ft[15..0 ];
* repeat subtract operation for 2nd and 3rd halfword *;
fd[63..48] ← fs[63..48] - ft[63..48];

PSUBW
fd[31..0 ] ← fs[31..0 ] - ft[31..0 ];
fd[63..32] ← fs[63..32] - ft[63..32];

PSUBD
fd[63..0] ← fs[63..0] - ft[63..0];
```

例外： 无。

C.2.25 PSUBSB/PSUBSH （有符号整数包减）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSUBSB 1 1 1 0 0	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSUBSH 1 1 0 0 0	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

格式：

```
PSUBSB fd,fs,ft
PSUBSH fd,fs,ft
```

功能： 有符号整数包减法指令。

描述： 这两条指令用于执行各种有符号整数包的减法操作：它们分别将第一个和第二个操作寄存器的对应位置的各个操作数（依照指令给定的数据类型）相加，并将结果存储在目标操作数中。溢出处理采用有符号饱和模式。这些指令的操作数皆为 64 位。

PSUBSB 指令的具体操作是将第一个和第二个操作寄存器对应的 8 个有符号字节整数相减。当任何一个减法结果太大或太小，超出了一个有符号字节的整数范围（即大于 7FH 或少于 80H），则对应的饱和值 7FH 或 80H，将被写入目的操作数。相应的，PSUBSH 指令操作有符号半字整数。当一个半字相减的结果超出了有符号半字整数范围时（即大于 7FFFH 或小于 8000H），则 7FFFH 或 8000H 的作为饱和值，将写入目标操作数。

操作：

```
PSUBSB
fd[ 7..0 ] ← SaturateToSignedByte(fs[ 7..0 ] - ft[ 7..0 ]) ;
* repeat subtract operation for 2nd through 7th bytes *;
fd[63..56] ← SaturateToSignedByte(fs[63..56] - ft[63..56]);

PSUBSH
fd[15..0 ] ← SaturateToSignedHalfword(fs[15..0 ] - ft[15..0 ]);
* repeat subtract operation for 2nd and 4th halfwords *;
fd[63..48] ← SaturateToSignedHalfword(fs[63..48] - ft[63..48]);
```

例外： 无。

C.2.26 PSUBUSB/PSUBUSH （无符号整数包减）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSUBUSB 1 1 1 0 1	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PSUBUSH 1 1 0 0 1	ft	fs	fd	SUB 0 0 0 0 0 1
6	5	5	5	5	6

格式：

```
PSUBUSB fd,fs,ft
PSUBUSH fd,fs,ft
```

功能： 无符号整数包减法运算。

描述： 这两条指令用于执行各种无符号整数包减操作：它们分别将第一个和第二个操作寄存器的对应位置的各个操作数（依照指令给定的数据类型）相减，并将结果存储在目标操作数中。溢出处理采用无符号饱和模式。

PSUBUSB 指令的具体操作是将第一个和第二个操作寄存器对应的 8 个无符号字节整数相减。当任何一个减法结果太小，超出了一个无符号字节的整数范围（即小于 00H），则 00H 将作为饱和值写入目的操作数。相应的，PSUBUSH 指令操作无符号半字整数。当一个半字相减的结果超出了无符号半字整数范围时（即小于 0000H），则 0000H 将作为饱和值将写入目标操作数。

操作：

```
PSUBUSB
fd[ 7..0 ] ← SaturateToUnsignedByte(fs[ 7..0 ] - ft[ 7..0 ]) ;
* repeat add operation for 2nd through 7th bytes *;
fd[63..56] ← SaturateToUnsignedByte(fs[63..56] - ft[63..56]);

PSUBUSH
fd[15..0 ] ← SaturateToUnsignedHalfword(fs[15..0 ] - ft[15..0 ]);
* repeat add operation for 2nd and 3rd halfwords *;
fd[63..48] ← SaturateToUnsignedHalfword(fs[63..48] - ft[63..48]);
```

例外： 无。

C.2.27 PUNPCKHBH/PUNPCKHHW/PUNPCKHWD （高位拆包）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PUNPCKHBH 1 1 0 1 1	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PUNPCKHHW 1 1 0 0 1	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PUNPCKHWD 1 1 1 0 1	ft	fs	fd	FLOOR.L 0 0 1 0 1 1
6	5	5	5	5	6

格式：

```
PUNPCKHBH fd,fs,ft
PUNPCKHHW fd,fs,ft
PUNPCKHWD fd,fs,ft
```

功能： 不同类型整数包的高位拆包操作。

描述： 这三条指令用于对不同类型的整数包的高位拆包。其具体操作为将第一个操作数和第二个操作数的高位（32 位）取出，并拆分成给定的宽度（PUNPCKHBH：字节；PUNPCKHHW：半字；PUNPCKHWD：字），然后按交替的顺序合并到目的寄存器中。所有低位信息被忽略。更具体的拆包后合并的位对应关系可参见操作部分。

如果第二个操作数值为 0，则这些指令的实质结果是分别将高位的四个字节转换为四个半字，高位的两个半字为两个字，高位的一个字为双字（都是无符号类型）。

操作：

```
PUNPCKHBH
fd[ 7..0 ] ← fs[39..32];
fd[15..8 ] ← ft[39..32];
fd[23..16] ← fs[47..0 ];
fd[31..24] ← ft[47..0 ];
fd[39..32] ← fs[55..48];
fd[47..0 ] ← ft[55..48];
fd[55..48] ← fs[63..56];
fd[63..56] ← ft[63..56];

PUNPCKHHW
fd[15..0 ] ← fs[47..32];
fd[31..16] ← ft[47..32];
fd[47..32] ← fs[63..48];
fd[63..48] ← ft[63..48];

PUNPCKHWD
fd[31..0 ] ← fs[63..32]
fd[63..32] ← ft[63..32];
```

例外： 无。

C.2.28 PUNPCKLBH/PUNPCKLHW/PUNPCKLWD （低位拆包）

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PUNPCKLBH 1 1 0 1 0	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PUNPCKLHW 1 1 0 0 0	ft	fs	fd	DIV 0 0 0 0 1 1
6	5	5	5	5	6

31–26	25–21	20–16	15–11	10–6	5–0
COP2 0 1 0 0 1 0	PUNPCKLWD 1 1 1 0 0	ft	fs	fd	FLOOR.L 0 0 1 0 1 1
6	5	5	5	5	6

格式：

```
PUNPCKLBH fd,fs,ft
PUNPCKLHW fd,fs,ft
PUNPCKLWD fd,fs,ft
```

功能： 不同类型整数包的低位拆包操作。

描述： 这三条指令用于对不同类型的整数包的低位拆包。其具体操作为将第一个操作数和第二个操作数的低位（32 位）取出，并拆分成给定的宽度（PUNPCKHBH：字节；PUNPCKHHW：半字；PUNPCKHWD：字），然后按交替的顺序合并到目的寄存器中。所有低位信息被忽略。更具体的拆包后合并的位对应关系可参见操作部分。

如果第二个操作数值为 0，则这些指令的实质结果是分别将低位的四个字节转换为四个半字，低位的两个半字为两个字，低位的一个字为双字（都是无符号类型）。

操作：

```
PUNPCKLBH
fd[63..56] ← ft[31..24];
fd[55..48] ← fs[31..24];
fd[47..0 ] ← ft[23..16];
fd[39..32] ← fs[23..16];
fd[31..24] ← ft[15..8 ];
fd[23..16] ← fs[15..8 ];
fd[15..8 ] ← ft[ 7..0 ];
fd[ 7..0 ] ← fs[ 7..0 ];

PUNPCKLHW
fd[63..48] ← ft[31..16];
fd[47..32] ← fs[31..16];
fd[31..16] ← ft[15..0 ];
fd[15..0 ] ← fs[15..0 ];

PUNPCKLWD
fd[63..32] ← ft[31..0 ];
fd[31..0 ] ← fs[31..0 ];
```

例外： 无。